Low-Energy Architectures of Linear Classifiers for IoT Applications using Incremental Precision and Multi-Level Classification

Sandhya Koteshwara
University of Minnesota, Twin Cities
Minneapolis, MN
kotes001@umn.edu

Keshab K. Parhi
University of Minnesota, Twin Cities
Minneapolis, MN
parhi@umn.edu

ABSTRACT

This paper presents a novel *incremental-precision classification* approach that leads to a reduction in energy consumption of linear classifiers for IoT applications. Features are first input to a low-precision classifier. If the classifier successfully classifies the sample, then the process terminates. Otherwise, the classification performance is incrementally improved by using a classifier of higher precision. This process is repeated until the classification is complete. The argument is that many samples can be classified using the low-precision classifier, leading to a reduction in energy. To achieve incremental-precision, a novel data-path decomposition is proposed to design of *fixed-width* adders and multipliers. These components improve the precision *without* recalculating the outputs, thus reducing energy. Using a linear classification example, it is shown that the proposed incremental-precision based multi-level classifier approach can reduce energy by about 41% while achieving comparable accuracies as that of a full-precision system. [1]

CCS CONCEPTS

• **Hardware → Arithmetic and datapath circuits**; **Application specific integrated circuits**;

KEYWORDS

Incremental-precision, Low-energy, Multi-level classification, Data-path decomposition, Fixed-width multiplication

1 INTRODUCTION

There is a growing trend towards connecting millions of devices to form what is termed as the Internet-of-Things (IoT). Reducing the energy consumption of these IoT devices connected to the network has become critical. Also, there is an increasing need to incorporate machine learning algorithms onto resource-constrained hardware to perform several tasks. Hence, low-energy solutions for IoT machine learning applications have become important. In this paper, we approach classification using bit-width reduction and a multi-level approach. This means that we compute features and perform classification using the lowest possible precision and then improve classification accuracy in a step by step manner based on certain criteria. The argument is that significant amount of data can be classified using the lowest precision components and only the remaining data need to be processed using higher precision components. Thus, overall energy consumption can be reduced.

The paper presents an incremental-precision algorithm for classification termed as *multi-level* classification. To process the data in each stage of the algorithm, we need hardware components which start processing in low precision and then increase precision as needed. This necessitates the design of basic components such as adders and multipliers which can perform the required *incremental-precision* operation. The required adders and multipliers are designed using *fixed width* components based on novel data-path decomposition techniques. Using multi-level classification implemented with incremental-precision components, we demonstrate energy reduction in a linear classification system.

The rest of the paper is divided as follows: Section 2 describes the incremental-precision addition and subtraction units. In Section 3, the incremental-precision multiplier based on approximation is introduced. The multi-level classification algorithm is presented in Section 4. Finally, using a linear classification example, results with respect to both accuracy and energy consumption are presented in Section 5.

2 INCREMENTAL-PRECISION ADDITION AND SUBTRACTION

Consider the sum of two N bit numbers A and B given by $S = A + B$. If we split A and B into their MSB and LSB parts, we obtain the sum in the following format:

$$S = A_{MSB} + 2^{-p}A_{LSB} + B_{MSB} + 2^{-p}B_{LSB}$$

In this equation, p represents the number of bits of the MSB. The sum can now be rewritten as the following:

$$\begin{aligned} S &= (A_{MSB} + B_{MSB}) + 2^{-p}(A_{LSB} + B_{LSB}) \\ &= S_M + 2^{-p}S_L \end{aligned} \tag{1}$$

This is termed as *data-path decomposition* and has been proposed in the context of error-tolerant applications [1].

The hardware architecture of a signed 8-bit incremental-precision adder/subtractor unit decomposed into two 4-bit adders/subtractors is illustrated in Figure 1. The top adder operates on 4 MSB bits of the data and stores it in memory. If a 4-bit output is desired, this output

[1]This research was supported in part by the National Science Foundation under grant number CCF-1749494.

GLSVLSI '18, May 23–25, 2018, Chicago, IL, USA

ACM ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194603

can be read. Instead, if an 8-bit output is desired, the sum of the MSB terms can be read from memory and combined with the output from the lower 4-bit adder/subtractor. The combination equations for both addition and subtraction are also presented in Figure 1. Note that the carry information from the lower adder needs to be included in the combination equation. Also since sign bits are considered, appropriate sign extensions are required for subtraction operation. The decomposition can be extended further on the upper or lower adder to create various combinations of incremental precision adders with precision 2-bit/4-bit/6-bit/8-bit. The decomposition of adder/subtraction units and combining the outputs of different precisions leads to an error-free result. However, this is not the case for computing multiplication using incremental-precision, as discussed in next section.

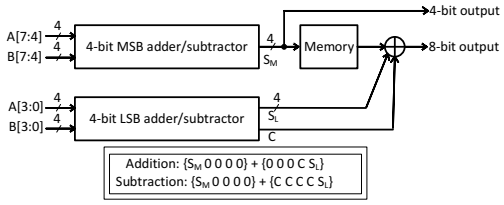


Figure 1: Decomposition and subsequent combination operations on 8-bit inputs to create incremental-precision adder/subtraction unit. The 8-bit inputs are decomposed into two 4-bit inputs in this example.

3 INCREMENTAL-PRECISION FIXED-WIDTH MULTIPLIERS

Consider the multiplication of two inputs x and y. Data-path decomposition can be applied on the multiplication operation of these two inputs as given by Equation (2).

$$P = x \times y = x_{MSB}\ y + 2^{-p}\ x_{LSB}\ y \tag{2}$$

x_{LSB} can be further decomposed to produce multiplication in the form of Equation (3), where p' is the number of bits in x_{LSB1}.

$$P = x_{MSB}\ y + 2^{-p}\ x_{LSB1}\ y + 2^{-(p+p')}\ x_{LSB2}\ y \tag{3}$$

This process can be continued on the LSB bits to produce a decomposed multiplier whose precision improves with every stage. The data flow of the decomposed multiplier is presented in Figure 2. To implement the products in this decomposition, fixed-width multipliers are required. Several approaches to implementation of fixed-width multipliers have been presented in the literature. Among them, the approach proposed in [2] is a technique which results in low-error implementations of fixed-width multipliers. This technique is described in detail in the next subsection. The following subsections discuss the modifications required to convert these fixed-width multipliers to *incremental-precision* fixed-width multipliers using certain approximations.

3.1 Fixed-width multipliers

Consider multiplication of two inputs x and y of length W bits. The resultant output after multiplication is a value P which is of width

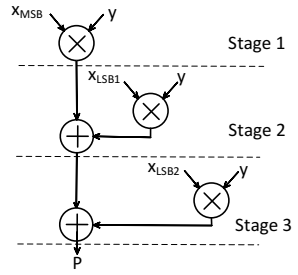


Figure 2: Data-path decomposition of multiplier into multiple stages by incrementally increasing the precision of x and maintaining y at a constant precision.

$2W-1$ bits. To perform multiplication, the coefficient y is first recoded into y' bits using Booth recoding techniques. This results in less number of partial products that need to be finally summed to generate the terms of the product. The fixed-width multiplier based on Booth recoding is illustrated in Figure 3.

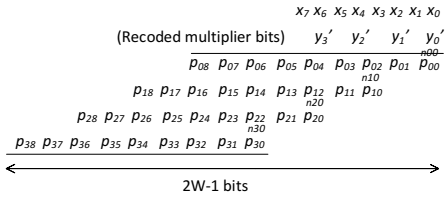


Figure 3: Multiplication of two 8-bit inputs by applying Booth recoding on the input y. The resultant output is of length $2W-1$ bits.

For a fixed-width multiplier, we constrain the number of terms in the final output to be of length W bits. If direct truncation is used, the resultant outputs produced will have high error. Hence, appropriate rounding needs to be performed and this idea is illustrated in Figure 4. In this figure, the final output is presented as three terms: *MP*, *LP_major* and *LP_minor* which represent the most significant part, the major part of the least significant part, and the minor part of the least significant part, respectively. The *MP* part needs to be computed precisely. This means that all the partial products required for this part need to be generated. The rest of the output can be approximated using an actual carry and approximate carry value whose details are discussed next. The final carry is termed as *error compensation bias*.

3.2 Approximate carry generation

The partial products of *LP_major* need to be computed precisely. For the terms of the *LP_minor*, statistical analysis can be used to derive an approximate carry value. Consider Table 1 which lists partial products for all possible combinations of recoding (represented by three adjacent bits of y) and combinations of adjacent x inputs. It can be seen that when the recoded value is greater than or equal to 1 (represented by $y'' = 1$), the expected value of partial products is $E[p_{i,j}] = 1/2$. Considering Figure 4, we see that the contribution

Table 1: Table of partial products and the expected values of partial products for every combination of recoded bits y' and input $x_j x_{j-1}$

					Partial products with $x_j x_{j-1}$ ($p_{i,j}$)				Expected value ($E[p_{i,j}]$)
y_{2i+1}	y_{2i}	y_{2i-1}	y'	y''	**00**	**01**	**10**	**11**	
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	1	1/2
0	1	0	1	1	0	0	1	1	1/2
0	1	1	2	1	0	1	0	1	1/2
1	0	0	-2	1	1	0	1	0	1/2
1	0	1	-1	1	1	1	0	0	1/2
1	1	0	-1	1	1	1	0	0	1/2
1	1	1	0	0	0	0	0	0	0

of each y'' bit on the expected value of *LP_minor* is 2^{-1}. Thus, the total expected value of *LP_minor* is given by :

$$E[LP\_minor] = 2^{-1}[y_0'' + y_1'' + y_2''] \tag{4}$$

The carry signals are generated depending on the values of y_0''

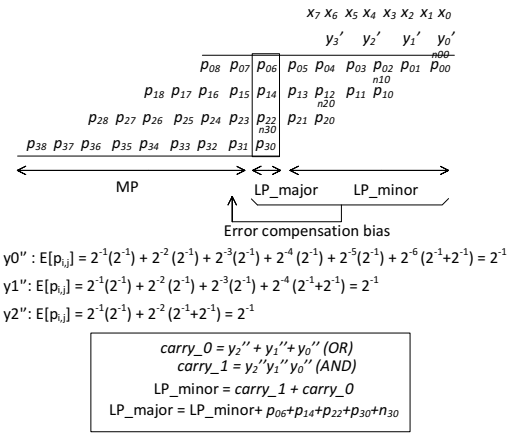


Figure 4: Calculation of the error compensation bias terms *LP_major* and *LP_minor* based on carry signals.

to y_2''. The maximum value of carry in this case occurs when all values of y_0'', y_1'' an y_2'' are equal to 1. This is equal to a value of $E[LP\_minor] = 1.5$ which can be rounded to a value of 2. Note that y_3'' does not contribute to the approximate carry value. For an expected value of 2, the number of carry signals required is 2. Solving for all values of y_0'' to y_2'' using Karnaugh maps, we obtain the required equations for the carry signals as shown in Figure 4.

3.3 Two-stage incremental-precision multipliers

From Equation (2), we require the computation of two products: $x_{MSB} \times y$ and $x_{LSB} \times y$. Observe that the coefficient y is of the same width while x is split into MSB and LSB parts. Hence, the previous example of 8×8 bit multiplication requires the decomposition of 8-bit operand x into two parts and the multiplication result is approximated. This can be achieved by separating the partial products and redesigning the carry generation circuit as illustrated in Figure 5. The top part of the figure presents the approximate multiplier

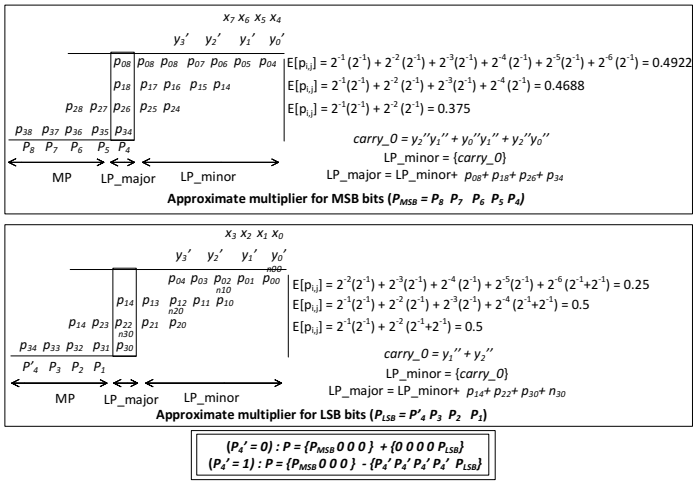


Figure 5: Splitting the 8-bit multiplier into two approximate multipliers of 4 bits each.

required for computation of the MSB bits. Note that the sign bit n_{t0}, where t denotes the row of the partial product, is not present in any of the partial product arrays. However, partial products p_{t8} is required in the calculations. The carry signal is also modified and reduced to just one compared to the 8-bit fixed-width multiplier presented in Figure 4.

The bottom part of the figure presents the approximate multiplier required for computation of LSB bits. In this case, the additional partial products (p_{t8}) are not present. However, the sign bit n_{t0} is processed by the bottom part. This requires the calculation of different carry signals. Note again that only one carry signal is required for this multiplier compared to two in the full precision fixed-width multiplier. The two data-paths can now operate *independently* of each other and produce approximate multiplication outputs. The combination of the two products is illustrated in Figure 5. The precision of the 4-bit multiplier operating on the MSB bits is improved by combining the product generated by operating on the LSB bits. The final output obtained through this combination will be an approximation of the full 8-bit precision fixed width output. This is because of the loss of precision during approximation of the products. Approximation is inherently introduced by the fixed-width multipliers as well as the incremental-precision architectures. It may be noted that approximations have been used in other contexts [3–5].

3.4 Multi-stage incremental-precision multipliers

The approximate multiplier required to compute the LSB bits can be further split into two as given by Equation (3). This idea is illustrated in Figure 6. The difference between the two multipliers is that the top multiplier does not account for the sign bit n_{t0} (similar to the

multiplier operating on MSB bits). The final output is a combination of the product from MSB bits overlapped with the product from the middle multiplier operating on the first two LSB bits and the last multiplier operating on the last two LSB bits. With each step, precision is improved.

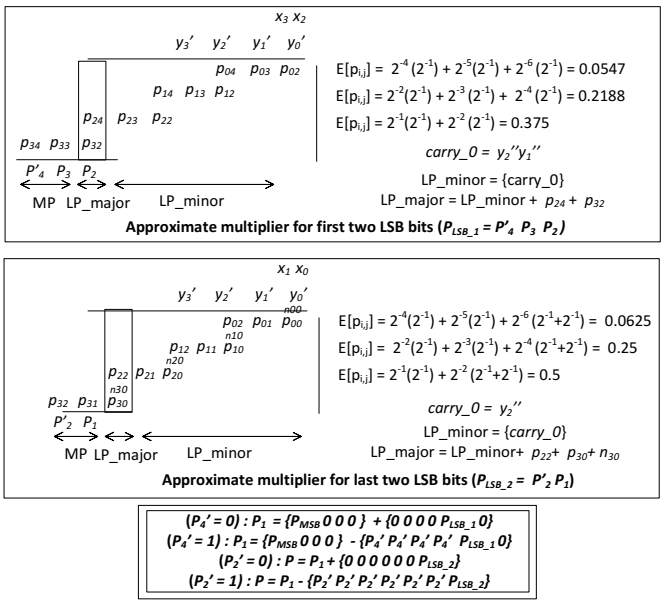


Figure 6: Splitting the LSB 4-bit multiplier into two approximate multipliers of 2 bits each.

We combine the three multipliers to produce a multi-stage incremental precision multiplier. Note that in the case of splitting an 8-bit multiplier into three stages of 4-bit, 2-bits and 2-bits, stage 2 and stage 3 of 2-bit multiplication can be performed using the same multiplier in a time-multiplexed or folded manner [6]. This idea is illustrated in Figure 7. The incremental-precision multiplier structure also requires a memory unit. The computation of intermediate stages are stored in memory.

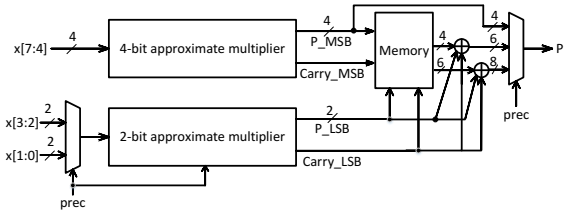


Figure 7: Incremental-precision 8-bit multiplier using one 4-bit multiplier and one time-multiplexed 2-bit multiplier. Intermediate outputs are stored in memory.

For example, if only a 4-bit precision output is demanded by the application, the MSB multiplier functions to generate a 4-bit output. The output along with a carry bit are stored in memory. If the application now demands a 6-bit output, the LSB multiplier is functional (with sign bit set to 0) and the memory unit is accessed to retrieve the MSB outputs. The combination of the two outputs is generated as the 6-bit output and also stored in memory along with the carry generated from the second stage. This process continues to provide incrementally higher precision outputs.

4 MULTI-LEVEL CLASSIFICATION ALGORITHM

The incremental-precision components described in this paper can be utilized in a multi-level classification setup where each stage of the classification is based on different precision levels for feature computation as well as classification. In this section, we describe a general multi-level algorithm with a specific example presented in later sections. The basic steps of the algorithm are illustrated in Figure 8.

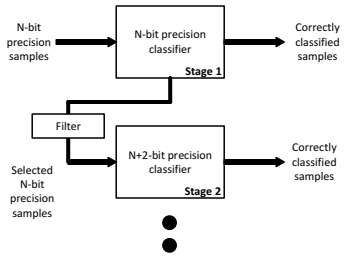


Figure 8: Steps of the multi-level classification algorithm with increasing precision classifier at each stage.

Training algorithm:

- Start with a low-precision classifier which operates at a precision of N-bits. Samples which are far away from the classification boundary can be easily classified with this classifier.
- Determine a *threshold* to filter out samples which need to be retrained in higher precision. This threshold could be selected based a distance metric which indicates the distance from the classification boundary.
- Using the determined threshold, select samples to be reclassified in the next higher precision of $N + l$-bits. Here l is the increment in precision between stages.
- This process can be continued till no further improvement in classification accuracy is obtained or for a predetermined number of steps.

Testing algorithm:

- For each test sample, use the first classifier of N-bit precision and obtain a distance metric or probability measure.
- Use the threshold determined in training process to select the samples which need to be reclassified.
- For the test samples which are reselected apply the next $N + l$-bit precision classifier.
- Continue till all stages of classifiers are applied or till no test sample is reselected.

We assume that training occurs offline and the testing/inference algorithm is implemented on hardware. For the first step of testing, a low-precision (N-bit) classifier is used. This implies that all computation can be carried out using the top-level of the decomposed

hardware (Figure 1 and Figure 7). If a sample is reselected to be classified at the next level, the bottom level of the decomposed hardware is used to increment the precision of the classification output. Note that it is not necessary to recompute the first N bits of the classification output. Since most of the samples can be classified using low-precision (N-bit) classifiers, the increment operation is utilized as needed. This leads to significant savings in energy consumption.

5 EXPERIMENTAL RESULTS

In this section we discuss a binary classification problem based on linear classifiers. Note that even though this is a simple example, the operations involved (namely multiplication with weights and addition of bias) are also fundamental to most machine learning algorithms such as logistic regression, linear Support Vector Machines (SVM), perceptron *etc.* Hence, the techniques in this paper can be applied to these algorithms as well after necessary extensions.

5.1 Multi-level linear classification

This example considers classification between samples of the Fisher's Iris data set [7]. Two classes of data belonging to two species are picked from this set. There are 50 samples in each category which are randomly divided into training and test data. 60% of the samples form the training set while 40% of the samples are considered for testing. For simplicity, only two features out of the four available features are considered. A linear classifier attempts to train weights to fit a straight line that separates the data. The classification problem can then be defined as the following:

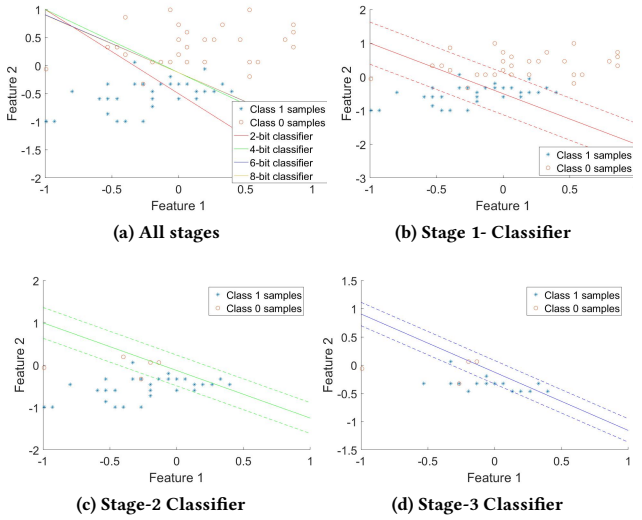
$$sign(w_1x_1 + w_2x_2 + w_0) > 0 \implies Class = 1 \tag{5}$$

In this equation, the features are represented as x_1 and x_2 while the weights are represented as w_0, w_1 and w_2. We observe that while the training process involves calculation of weights, the testing process involves multiplication and addition operations. Hence, we incorporate the incremental-precision components in this classification system.

Consider the multi-level classification algorithm described in the previous section with a starting precision $N = 2$ bits and an increment $l = 2$ bits. The plots of the data with the separating line in different precisions are presented in Figure 9. Note that this figure also presents a margin above and below the separating line which represents the *threshold* calculated in the training process. All samples lying beyond this threshold need not be retrained. The calculation of the threshold value is obtained after computing the value $w_1x_1 + w_2x_2 + w_0$ for each sample and setting the threshold equal to the average of the magnitudes of the computed value for all training samples. Note that other techniques for threshold calculations can be used with associated trade-offs.

5.2 Accuracy of classification

The results of training using a full-precision (8-bit) classifier and the multi-level classification algorithm are presented in Table 2. The table also provides the number of samples reselected at each stage. For the multi-level classification algorithm, the class labels for the samples selected at each stage are replaced with new class



(a) All stages (b) Stage 1- Classifier

(c) Stage-2 Classifier (d) Stage-3 Classifier

Figure 9: Classification of samples using multi-level classification algorithm.

Table 2: Accuracy of classification for the training and testing process using full-precision single stage and incremental-precision multi-level classifier

Classifier	Training accuracy	Samples trained	Testing accuracy	Samples tested
Full-prec (8-bit)	93.33%	100%	95%	100%
Stage 1 (2-bit)	50 %	100%	50%	100%
Stage 2 (4-bit)	93.33%	56.67%	90%	77.50%
Stage 3 (6-bit)	95%	31.67%	95%	20%
Stage 4 (8-bit)	95%	21.67%	95%	12.5%

labels and the classification accuracy for the entire training set is calculated. We observe that the classification accuracy improves with each stage and the number of samples reselected at each step also reduces.

Using the generated classifier models and the calculated thresholds from the training process, we generate the test accuracy at each stage of the algorithm. The test accuracy measures along with the number of reselected samples are also presented in Table 2. Stage-4 does not provide any improvement in accuracy and hence the algorithm can be stopped after 3 stages. These results are obtained using an assumption that the components used to implement the testing algorithm provide accurate precision values. In the next section we provide practical implementation results along with the reduction in energy consumption after employing incremental-precision components in hardware.

5.3 Test accuracy and reduction in energy using incremental-precision components

The classifier using both single stage and multi-level algorithm is implemented in Verilog Hardware Description Language and synthesized using Design Compiler with a 65 nm technology library.

Table 3: Accuracy of the classifier based on implementations of the full-precision and multi-level algorithms

Classifier implemented in hardware	With original thresholds		After recalculation of thresholds	
	Testing accuracy	Samples tested	Testing accuracy	Samples tested
Full-precision (8-bit)	92.5%	100%	92.5%	100%
Stage-1 (2-bit)	37.5%	100%	37.5%	100%
Stage-2 (4-bit)	72.5%	72.5%	85%	100%
Stage-3 (6-bit)	77.50%	17.5%	92.5%	35%

Table 4: Area, Power, Timing and Energy consumption per test sample for the single stage full-precision and multi-level incremental precision algorithms (Numbers in paranthesis indicate values after recalculation of threshold)

Classifier	Without time-multiplexing				With time-multiplexing			
	Area (um^2)	Total Power (mW)	Total time (ns)	Energy per test sample (pJ)	Area (um^2)	Total Power (uW)	Total Time (ns)	Energy per test sample (pJ)
Full-precision (8-bit)	1123	1.480	167	0.155	1123	1.48	167	0.155
Stage-1 (2-bit)	-	0.284 (0.284)	181 (181)	-	-	0.284 (0.284)	181 (181)	-
Stage-2 (4-bit)	-	0.234 (0.323)	150 (207)	-	-	0.249 (0.344)	151 (209)	-
Stage-3 (6-bit)	-	0.071 (0.143)	25 (50)	-	-	0.073 (0.145)	26 (53)	-
Incremental-precision	987	0.589 (0.75)	356 (438)	0.068 (0.086)	867	0.606 (0.773)	358 (443)	0.072 (0.091)

All implementations are clocked at a frequency of 100 MHz and performance is measured in terms of area, power and time required for testing all samples as well as the energy required per sample.

The testing accuracy obtained is reported in Table 3. Note that with a full-precision system, an accuracy of 92.5% can be obtained. This is lower than the value obtained through simulation and can be attributed to errors in the multiplication hardware. For the incremental-precision system, we observe that using the original thresholds calculated through simulation, high test accuracies cannot be obtained. This is due to lack of accounting for the additional approximation of the incremental-precision components. After recalculation of the thresholds, the incremental-precision system is able to achieve the same accuracy as that of the fixed-width full-precision classification system.

Performance measures as well as resource consumption after synthesis are presented in Table 4. While the area consumption is not dependent on the number of samples processed at each step, the power consumption and timing requirements are presented based on the number of samples processed. We present the energy consumption per sample after summing the energy requirements for each stage (obtained as a product of the power and timing requirement) and then dividing by the total number of test samples. Implementations both with time-multiplexing of the lower stage and without time-multiplexing are discussed. For each implementation, the values before and after recalculation of thresholds are also presented.

We observe that the energy consumption for the incremental-precision system with the original thresholds is reduced by about 56% compared to the full-precision system. After recalculation of thresholds, the energy reduction is about 45%. With the multiplexed hardware, these values correspond to 53% and 41%, respectively. While the implementation without multiplexing reduces the area of the incremental-precision system by 12%, multiplexing further reduces the area consumption by 22%. The improvements in energy and area consumption are achieved at an expense of increase in the time to process. This is because of the overheads associated with comparison and combination. This timing overhead can be reduced by use of pipelining techniques such that multiple samples can be processed in parallel [6].

6 CONCLUSIONS

This paper presents a novel multi-level classification algorithm and corresponding incremental-precision adders and multipliers that can be used to implement it. Note that the incremental-precision idea is not limited to the classifiers and can be applied to the features as well. Also, the proposed algorithms and architectures are not limited to linear classification and can be used in several machine algorithms with more sophisticated classifiers. The corresponding error-energy trade-off needs to be carefully considered.

REFERENCES

[1] Sai Zhang and Naresh R Shanbhag. 2016. Embedded Algorithmic Noise-Tolerance for Signal Processing and Machine Learning Systems via Data Path Decomposition. *IEEE Transactions on Signal Processing* 64, 13 (2016), 3338–3350.

[2] Kyung-Ju Cho, Kwang-Chul Lee, Jin-Gyun Chung, and Keshab K Parhi. 2004. Design of low-error fixed-width modified booth multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12, 5 (2004), 522–531.

[3] S Hamid Nawab, Alan V Oppenheim, Anantha P Chandrakasan, Joseph M Winograd, and Jeffrey T Ludwig. 1997. Approximate signal processing. *Journal of VLSI signal processing systems for signal, image and video technology* 15, 1-2 (1997), 177–200.

[4] Jongsun Park, Jung Hwan Choi, and Kaushik Roy. 2010. Dynamic bit-width adaptation in DCT: an approach to trade off image quality and computation energy. *IEEE transactions on very large scale integration (VLSI) systems* 18, 5 (2010), 787–793.

[5] Charbel Sakr, Ameya Patil, Sai Zhang, Yongjune Kim, and Naresh Shanbhag. 2016. Understanding the Energy and Precision Requirements for Online Learning. *arXiv preprint arXiv:1607.00669* (2016).

[6] Keshab K Parhi. 1999. VLSI digital signal processing systems: design and implementation. (1999).

[7] Ronald A Fisher. 1936. The use of multiple measurements in taxonomic problems. *Annals of human genetics* 7, 2 (1936), 179–188.