Dynamic Approximation of JPEG Hardware

Farhana Sharmin Snigdha, Deepashree Sengupta, Jiang Hu, and Sachin S. Sapatnekar

***Abstract*—JPEG compression based on the discrete cosine transform (DCT) is a key building block in low-power multimedia applications. Approximate computation techniques are used to exploit the error tolerance of JPEG. An image-dependent framework is proposed in this work to design optimized approximate hardware with variable approximate bit-widths for a user-specified error budget. The proposed method can dynamically adjust the extent of approximation in the system depending on the pixel values of the input image, thus leveraging the inherent sparsity of certain images. This novel technique not only improves the power-delay product by 3.4× over the base case, i.e., where the JPEG hardware is accurate, but also significantly outperforms the image-independent approximation case, which is solely based on the error tolerance of the JPEG algorithm.**

***Index Terms*—Approximate computing; Low-power design; JPEG; Image compression; Image contrast level; Image-dependent approximation; Image-independent approximation; Nonlinear optimization**

I. Introduction

Energy-efficiency demands have driven interest in designs that leverage the inherent error resilience of multimedia applications [1]–[4]. In this context, approximate computing uses simplified, lower power hardware operations with controlled errors, using logic or architecture modification, voltage over-scaling, or time starvation. A typical target for this optimization is in the domain of digital signal processing (DSP) for multimedia. Typical DSP operations used in image and video processing units include image compression, filtering, and reconstruction. We develop an optimized approximate computing solution for JPEG-based image compression.

A core building block for JPEG compression is the discrete cosine transform (DCT). Several algorithms for implementing the DCT have been proposed to reduce the number of computations without approximations by altering the architecture or/and logic simplifications [5]–[7] by exploiting the periodic symmetries of cosine terms in the DCT flow graph. However, even for a moderate-size image, the number of computations is large [8]. A few prior methods have explored approximations in JPEG compression, but perform *ad hoc* optimizations. Some approaches use dynamic bit width [9] and dynamic range reduction [3], while others uniformly approximate all adders in the DCT [1]. Other approaches [10], [11] target general RTL structures and do not exploit the structure of JPEG.

We address the problem of optimizing JPEG compression hardware by approximating the arithmetic units within the JPEG block to reduce the power consumption under a user-specified error budget. This is a worthwhile target for optimization for two reasons. First, JPEG compression is used widely enough for the results to be useful, and second, the concepts developed here could potentially be extended to optimize other structures such as FFTs, MPEGs.

We propose two approaches to implement approximations in the JPEG hardware. The first approach, presented in [12], is an image-independent static approximation that considers the sensitivity of the error at the output for the approximations at the arithmetic units within the block. An optimization problem is solved to maximize power savings while ensuring that the output error lies within a user-specified budget. An analytical formulation to select the approximate bits for the arithmetic units is proposed in [13]. However, such a scheme is necessarily conservative since it provides a single approximation scheme that is safe for all input images. The differences in the pixel patterns of images imply that the distributions of input values differ from image to image, and the level of approximation can be modified to be image-specific. Our second approach reduces the conservatism of the first approach by tailoring the level of approximation to the input image. We propose an input dependent dynamic approximation scheme that is implemented over the static approximation process. A few prior works have explored input-dependent approximations without characterizing the input pixel variations [14], [15]. In our work, the input distributions of different images are estimated in real time to determine the level of approximations. Run-time dynamic approximation is enabled by the addition of decision circuitry.

In Sections II and III, we discuss the principles of JPEG compression and its approximate hardware framework, respectively. The ideas of static and dynamic approximation scheme are explained, respectively, in Section IV and V. Next, Section VI contains the more detailed explanations of dynamic approximation process and heuristics to simplify the architecture. Section VII compares the power, area, delay and quality for various error targets and images using our framework and we conclude in Section VIII.

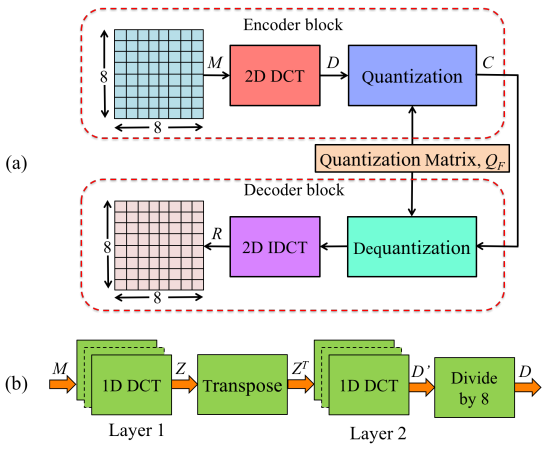


Fig. 1: (a) Block diagram of a JPEG encoder and decoder and (b) decomposing a 2D DCT block into 1D DCTs.

II. JPEG Decomposition

As illustrated in Fig. 1(a), the JPEG architecture is divided into the encoding and decoding phases. The JPEG compression scheme in the encoder block is based on a 2-dimensional (2D) frequency-domain DCT, followed by quantization. The subscript F of the quantization matrix, Q_F, is a quality factor ranging from 1 to 100, where 1 corresponds to the lowest quality. Commonly-used values of F are 50 and 90; a lower quality factor, F, achieves a higher compression ratio [16]. The method operates by dividing an image, I_{image}, into 8×8 pixel blocks and compressing each block separately. Each such block constitutes the 8×8 initial matrix, I, which will be referred to as one frame, f. For example, a typical image of size 512×512 pixel consists of $N_f = 4096$ frames. Each element of the matrix, I, has a value in the range of 0 to 255, but the DCT operation is performed on a translated matrix, M, obtained from I by subtracting 128 from each entry in I. The result of compressing M is an 8×8 matrix, C. The matrix C is used to create a reconstructed image matrix, R, through the JPEG decoder block. Decoding is analogous to compression, except that the inverse DCT (IDCT) replaces the DCT.

The separable property of the DCT allows the 2D operation to be decomposed in two sequential 1D DCT operations along the rows and columns of M, as shown in Fig. 1(b). We denote the first and second sets of 1D DCT blocks as Layer 1 and Layer 2, respectively, where each layer contains eight individual 1D DCT blocks. A number of fast 1D DCT algorithms have been proposed [5]–[7] to exploit the periodic nature of cosine terms in 2D DCT equations [12]. We have chosen Loeffler 1D DCT algorithm [7] as the basis for our work, as it achieves the theoretical lower bound for multiplications. The Loeffler 1D DCT can be represented as the directed acyclic graph (DAG) structure shown in Fig. 2. Each node in the DAG represents arithmetic operations such as add/subtract or multiply. There are total of eight stages, $s(i), i = 1, \cdots, 8$, between the input and output. There are 11 multiplications and 29 additions in each 1D DCT block. The net computation over the 16 1D DCT blocks in the 2D DCT block requires 176 multiplications and 464 additions per frame.

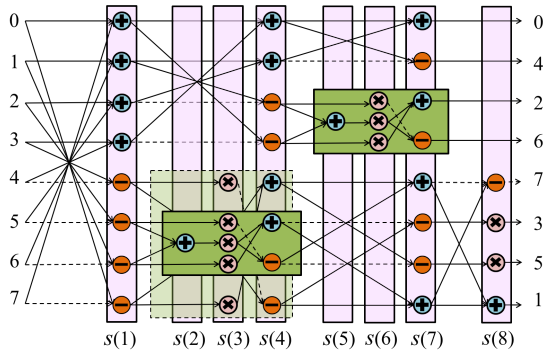


Fig. 2: A DAG model of the Loeffler DCT.

The DCT architecture varies the computations bit-widths with the dynamic range of the node. All calculations are performed using variable bit-width signed arithmetic. It is important to note that all multipliers perform constant multiplications, where the fractional constant inputs are stored as fixed-bit integers by multiplying them by 2^7; this scaling is reversed at the output through shift operations.

III. Approximate Computing in JPEG

The amount of computation and power dissipation in a DCT is significant: for a moderate-sized 512×512 (256K) image with 4096 frames, the fast DCT requires nearly 10^6 multiplications and 2×10^6 additions. This, along with the error resilience of JPEG, motivates us to explore approximate computing.

We explore the use of approximation in the DCT block. We evaluate the quality of the solution by uncompressing the approximate JPEG image using exact computations in the IDCT block (the same ideas could be applied to the IDCT block). The adders and multipliers are implemented as arrays of full adders (FAs). Let us assume that N_{a_x} and N_{m_y} are the number of bits corresponding to the dynamic range of data computed in the adder node, a_x, and the multiplier node, m_y, respectively. In our study, we implement addition using N_{a_x}-bit ripple carry adders and multiplication using a $4:2$ compressed carry save array (CSA) based N_{m_y}-bit lower triangle multiplier.

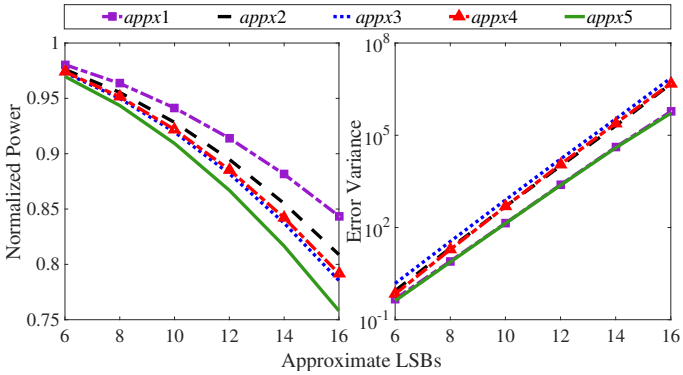


Fig. 3: Power and error variance comparison for various approximate multipliers.

To build approximate implementations, some of the exact FAs that implement each arithmetic operation can be replaced by their approximate counterparts. In particular, for each operation, a certain number of the least significant bits (LSB) can be approximated for each node in the DAG. The approximate FA could correspond to, for example, one of the five transistor-level FA designs proposed in [1]. In our work, we narrow this down further by evaluating the power dissipation and error variance for these five FA designs. The evaluation proceeds by performing 5000 Monte Carlo simulations on a 32-bit multiplier for two independent Gaussian inputs with various approximate LSBs utilizing the FAs from [1]. For various numbers of approximated bits, the power requirements and corresponding error variance for the five FA designs, as compared to the accurate multiplier, are shown in Fig. 3. The area and delay improvements show the same trends as power and are not shown. For all levels of approximations, it can be seen that the *appx5* design provides the lowest error levels and power relative to the other options.

The JPEG compression error depends on the approximation used at each node. We explore two classes of approximations: (1) *Static* approximation chooses the degree of approximation for the nodes once during system optimization, and maintains the same approximation for all input images. The choice must be conservative to function correctly for any input image. (2) *Dynamic* approximation depends on the data in a specific image, and generates image-dependent approximations chosen on the fly, depending of the error-resilience of the image.

IV. Static Approximation

We now develop a static optimization framework to optimize the approximation level for each computational unit in the Loeffler network, based on models that relate the introduced error to the level of approximation. The optimization is carried out over the two layers in Fig. 1(b), each containing eight 1D Loeffler DCT structures, each of which processes one column of the M matrix. The full set of optimization variables corresponds to the approximation levels in the 29 adders and 11 multipliers in each of the 8 blocks and in both levels of the 2D DCT. We denote the full list of adder and multiplier optimization variables by the sets θ_a and θ_m, respectively. This leads to a total of $(29+11)\times 8\times 2 = 640$ variables. However, such an optimization could imply that each 1D DCT block could have a different structure since it may be optimized differently. Since this surrenders the major advantage of regularity that is exploited in layout optimization of DSP blocks, we choose to restrict the optimization so that within each layer, all eight 1D DCT blocks are identical, allowing for easier design and layout. This leads to a reduced number of variables, θ'_a and θ'_m, so that the total number of variables is now $(29+11)\times 2 = 80$, thus also reducing the size of the optimization problem.

We consider the impact of using approximate multipliers on the accumulated error at the output of the JPEG compression engine, and its effect on the quality of the resultant image. The error at each node within the DAG has a different contribution to the total error at the final output. We propose a scheme based on the output error sensitivity of each node to optimize the hardware, subject to a user-specified error budget that is typically application-dependent. We proceed in three steps:

- **Computing the error statistics of each unit module:** We characterize the error mean and variance for arithmetic units (adders and multipliers) under static approximation criteria. We assume that the input patterns are uniformly distributed, and this is reasonable over a large number of frames.
- **Sensitivity analysis of the DCT block:** We calculate the sensitivity of each node in a 1D DCT block to the error at the output stage, as an input to the optimization engine.
- **Nonlinear optimization:** We formulate a nonlinear optimization problem that is solved to obtain the optimal number of approximated bits for each adder and multiplier.

A. Computing Error Statistics

Errors in Adders: For the approximate FA design *appx5*, the error x from approximating α LSBs can take on 2^α different values from the set $\{0, \pm 1, \pm 2, \cdots, \pm\left(2^{\alpha-1}-1\right), -2^{\alpha-1}\}$. The approximation scheme in the adder is concentrated in the lower LSBs. Over the large number of frames processed by an adder, the lower LSBs of the input have equal probability of ones and zeros. Using this uniform distribution of error values, i.e., $p_x = 1/2^\alpha$, we find the error mean and variance:

$$\mu_a(\alpha) = -0.5; \qquad \sigma_a^2(\alpha) = \left(4^\alpha - 1\right)/12 \tag{1}$$

The mean is independent of number of approximated bits. Even for a 1 LSB approximation, 3σ is 0.75 units and exceeds the mean. The variance is larger for more approximated bits, and the mean can be reasonably approximated as zero.

Errors in Multipliers: In the JPEG computation, each multiplier has one constant and one variable node, as described in Section II. Assuming a uniform input distribution, the error statistics of multipliers are analytically modeled using the MATLAB curve fitting toolbox as a function of the α approximated LSBs as:

$$\sigma_m^2(\alpha) = 0.6135(4^\alpha) \tag{2}$$

As before, the error mean is essentially zero.

B. Sensitivity Analysis of the DCT Block

The computation at each node of a 1D DCT block is a linear combination of the results of its immediate predecessor stage node. If we denote the result at node n of stage s, as $T_{n,s}$,

$$T_{n,s} = \textstyle\sum_{k=1}^{8} W_{k,n,(s-1)} \times T_{k,(s-1)}, \quad 1 \le n, s \le 8 \tag{3}$$

where $W_{k,n,(s-1)}$ is the weight of node k of the previous stage, $(s-1)$ for node n of stage s. The value of $W_{k,n,(s-1)}$ can be obtained from the structure of the network, shown in Fig. 2. We use $\zeta_{n,s,k}$ to denote the sensitivity of node k of stage $(s-1)$, at node n of stage s. From (3),

$$\zeta_{n,s,k} = \frac{\partial T_{n,s}}{\partial T_{k,(s-1)}} = W_{k,n,(s-1)} \qquad \forall n,s,k \in \{1,\cdots,8\} \tag{4}$$

The error variance at a node can be obtained by weighting the variances of predecessor nodes by the sensitivity, and adding the error generated due to an approximation at the node. The error variance, $\sigma^2_{n,s,b,l}$, for node n, stage s, block b, layer l is:

$$\sigma^2_{n,s,b,l} = \textstyle\sum_{k=1}^{8} \zeta^2_{n,s,k} \times \sigma^2_{k,(s-1),b,l} + \sigma^2_{op,nsl} \tag{5}$$

where

$\sigma^2_{op,nsl}$ = Generated variance, node n, stage s, layer l

$\sigma^2_{n,0,b,l}$ = Input variance for block, b, of layer, l

The values of $\sigma^2_{op,nsl}$ can be obtained from (1) or (2). We propagate the error variance up to the D' node of Fig. 1(b) using (5) and match it with the error variance budget, $\sigma^2_{budget,D'}$. For a quality factor, F, and a maximum user-specified error budget, δ_R, an empirical relation between $\sigma^2_{budget,D'}$ and $\sigma^2_{budget,R}$ at node R in Fig. 1(a), is as follows:

$$\sigma^2_{budget,D'} = \tau \times \sigma^2_{budget,R} \tag{6}$$

$$\text{where} \qquad \tau = \frac{F}{100} \times \left[-2.94 \times 10^5 \delta_R^{-2.25} + 250\right]$$

This formula captures the factors that contribute to the difference between $\sigma^2_{budget,D'}$ and $\sigma^2_{budget,R}$, namely, (a) a lower

F incurs a larger quality degradation in quantization, and (b) the inputs to the DCT nodes in Fig. 2 are not independent, as assumed, due to correlations.

To obtain $\sigma^2_{budget,R}$, we begin with the maximum error δ_R, specified by the user at the 3σ point of the error at R, based on the application requirement. This can be written as

$$\delta_R = \mu_{budget,R} + 3\sigma_{budget,R} \approx 3\sigma_{budget,R} \tag{7}$$

where $\mu_{budget,R} \approx 0$ and $\sigma_{budget,R}$ are the mean and standard deviation of the maximum error budget, respectively. Thus,

$$\sigma^2_{budget,R} = (\delta_R/3)^2 \tag{8}$$

C. The Nonlinear Optimization Formulation

For a reasonable k approximate output LSBs in an array multiplier, the number of FAs is quadratic in k. For an adder, each approximate output bit translates to one approximate FA. Thus, the number of approximate FAs in an adder, $f_a(k)$, and and multiplier, $f_m(k)$, are given by

$$f_a(k) = k \; ; \; f_m(k) = k(k-1)/2 \tag{9}$$

Therefore, the total number of FAs that are approximated can be represented as a function of the number of approximated bits associated with each adder or multiplier, given by:

$$\lambda_{appx} = 8 \times \left(\sum_{a_x \in \theta'_a} f_a(\alpha_{a_x}) + \sum_{m_y \in \theta'_m} f_m(\alpha_{m_y}) \right) \tag{10}$$

Here, θ'_a and θ'_m correspond to the reduced sets of adders, a_x, and multipliers, m_y, for the 2D DCT block. As discussed earlier, the same approximation is used within each of the 8 1D DCT blocks in layer, $l = 1, 2$, to preserve layout regularity. The multiplicative factor of 8 captures this notion.

Based on the relations in Sections IV-A and IV-B, we formulate an optimization problem that maximizes the savings due to approximation, subject to a specified bound on the error introduced at D' node for Fig. 1(b). The precise formulation of the optimization problem is:

$$\begin{aligned}
\max \quad & \sum_{a_x \in \theta'_a} \alpha_{a_x} + \sum_{m_y \in \theta'_m} \frac{\alpha_{m_y}(\alpha_{m_y}-1)}{2} && (11)\\
\text{s. t.} \quad & \text{(a) } \forall n, s, b \in \{1, \cdots, 8\}, l \in \{1, 2\}, \\
& \qquad \sigma^2_{n,s,b,l} = \sum_{k=1}^{8} \zeta^2_{n,s,k} \times \sigma^2_{k,(s-1),b,1} + \sigma^2_{op,nsl} \\
& \text{(b) } \sigma^2_{n,0,b,1} = 0 \;\; \forall n, b \in \{1, \cdots, 8\} \\
& \text{(c) } \sigma^2_{n,0,b,2} = \sigma^2_{n,8,b,1} \;\; \forall n, b \in \{1, \cdots, 8\} \\
& \text{(d) } \sigma^2_{n,8,b,2} \leq \sigma^2_{budget,D'} \;\; \forall n, b \in \{1, \cdots, 8\}
\end{aligned}$$

The objective function maximizes λ_{appx}, representing the total hardware savings. The constraints represent limits on the error variances at every stage. Constraint (a) is the relationship (5) between the error variance from one stage to the next in each layer of the network. Constraints (b) and (c), respectively, state that there is no error at the input of layer 1 of the DCT on the compression side when M is presented, and that the error of stage 8 of layer 1 is passed on to layer 2. The variance budget constraint (d) is obtained from (7) and (8).

Since the error variance equations of adder and multiplier are nonlinear, as shown in (1) and (2), both the objective and constraint functions are nonlinear. The decision variables, α_{a_x} and α_{m_y}, for this optimization are integer-valued. Although a mixed integer linear programming problem can be computationally expensive, (a) the number of variables is small enough that a solution is realistic, and (b) since this is an one-time solution at design time, computation time is not critical. In practice, we find that the problem is solved in about 1 hour.

V. Dynamic Approximation

Under static optimization, the level of approximation is chosen to be safe for any input image. The error variance equations (1) and (2) assume uniform input probabilities for all adders and multipliers, respectively, but in practice, this approximation provides pessimistic error estimates. Using image-specific input statistics, dynamic approximation can increase the approximation level significantly. We present a real-time image-dependent computation process that dynamically finds the approximation at each node. Our starting point is the static approximation, which works for all images, and we extract additional gains through selective dynamic approximation.

A. Selecting Nodes for Dynamic Approximation

The DCT block has two node types: adders (or subtractors) and multipliers. Considerations for approximation include:

- **Number of image-dependent inputs at the node:** The adder nodes have two image-dependent inputs, while multiplier nodes have one constant and one image-dependent input. Therefore, the data-dependent approximation circuitry is required on both adder inputs, but only one multiplier input, resulting in lower hardware overhead for multipliers.
- **Node error sensitivity to the outputs:** In the DCT DAG, adder nodes are followed by constant multiplier nodes. Thus, an error introduced at the adder node is amplified. No such amplification occurs for approximations in multiplier nodes. As a result, the error sensitivities of the multiplier nodes are always smaller than those of the adder nodes.
- **Number of dynamic FA blocks at the node:** To achieve the maximum dynamic savings within the limited error budget, the number of approximated FA bits must be maximized. According to (9), incrementing the approximation level by one bit for an adder and a multiplier implies approximating $f_a(k+1) - f_a(k) = 1, f_m(k+1) - f_m(k) = k$ FAs, respectively. Therefore, more FAs are available, per bit of approximation, for multipliers than adders.

All of these factors imply that the dynamic approximation of multiplier nodes is more effective than adder nodes.

We now choose an approximate FA design for dynamic approximation, choosing from *appx1–appx5* [1]. We find that *appx5* is the best choice for dynamic approximation because:

- From Fig. 3, *appx5* is the most power-efficient option.
- The outputs of *appx5* only depend on the inputs a and b, not on the carry-in bit, c_{in} [1]. Therefore, the scope of propagated errors is limited since an error generated at the carry-out at a node is not sent to the next bit.
- The *appx5* design provides accurate results for two crucial combinations, $(a, b, c_{in}) = (000)$ and (111). This is particularly important because many inputs do not use the entire

bit width, but use sign-extension for their most significant bits (MSBs). The correct computation of these two combinations ensures that no errors are generated in processing sign-extension MSBs. The computation of multiplication involves taking the summation of the partial products after proper shift operation. If we only take summations of non-zero partial products, the sign extensions of all non-zero partial products are the same and thus according to the property of *appx5*, no error will be generated. If Booth's multiplication is used (which is not the case in this paper), then the signs will be mixed and this property will not hold.

Let us consider two n-bit multiplier inputs where the lower x and y bits, respectively, contain useful data, and the MSBs are sign-extension bits. If we approximate $\alpha \geq x+y$ bits using *appx5*, the error in the multiplier output is limited to an $(x+y)$-bit approximation, rather than an α-bit approximation, since the MSBs are correctly computed. For $\alpha < x+y$, the maximum error corresponds to α bits. In contrast, for *appx1–appx4*, the correctness of sign-extension does not hold, either because an error is generated for these combinations and/or because an error in an input carry can create an error in the sign-extension MSBs, so that the maximum error corresponds to α bits regardless of $x+y$. This is summarized in Table I.

TABLE I: Error characteristics of approximate multipliers

	Number of potentially erroneous bits	
	appx5 based multiplier	*appx1–appx4* based multiplier
$\alpha \geq x+y$	$x+y$	α
$\alpha < x+y$	α	α

B. Modification of Multiplier Error Variance Formulation

For a multiplier node, m_i, in Fig. 4, let y_i be its output, c_i and v_i^f its constant input and its variable input in frame f, respectively, α_{m_i} the number of bits used for static approximation, and $\sigma_m^2(\alpha_{m_i})$ its error variance, obtained from (2). When the variable input of the *appx5* based multiplier satisfies

$$\alpha_{m_i} \geq \log_2(|v_i^f|) + \log_2(c_i), \tag{12}$$

its error characteristics, shown in Table I, reveal two insights:

- Full approximation of all bits of the multiplier keeps the error unchanged, while saving considerable power.
- The true error variance, $\sigma_{m,true}^2(\alpha_{m_i})$, due to the static approximation is smaller than $\sigma_m^2(\alpha_{m_i})$, and, for both the static and dynamic cases, is given by $\sigma_m^2(\lceil \log_2(|v_i^f|) + \log_2(c_i) \rceil)$.

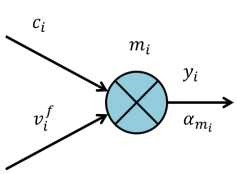
We relax the criterion of (12) to define the bound $\mathcal{N}_i$:

$$\alpha_{m_i} \geq \lceil \log_2(|v_i^f|) \rceil + \lceil \log_2(c_i) \rceil \tag{13}$$

$$\text{i.e., } \lceil \log_2(|v_i^f|) \rceil \leq \mathcal{N}_i = (\alpha_{m_i} - \lceil \log_2(c_i) \rceil) \tag{14}$$

to obtain a criterion based purely on v_i^f.

Let us assume, α_{m_i} and $z_i \in \mathbb{Z}_{\geq 0}$ are the static and additional dynamic approximate bits, respectively, for the multiplier node m_i. We can model the true introduced error variance, $\sigma_{m,true}^2(\alpha_{m_i}+z_i)$, as the weighted sum of the error variances of various approximate bits, where the weights are the probabilities of various input ranges:



Multiplier node	m_i
Constant input	c_i
Variable input	v_i^f
Multiplier output	y_i
Static approximation bit-width	α_{m_i}

Fig. 4: A typical multiplier processing data in frame f.

$$\sigma_{m,true}^2(\alpha_{m_i}+z_i) = \gamma_1(z_i) + \gamma_2(z_i) \tag{15}$$

where $\gamma_1(z_i) = \sum_{k=1}^{\mathcal{N}_i+z_i-1} P_k \sigma_m^2(k + \lceil \log_2(c_i) \rceil)$

$$\gamma_2(z_i) = \left[1 - \sum_{k=0}^{\mathcal{N}_i+z_i-1} P_k\right] \sigma_m^2(\alpha_{m_i}+z_i)$$

$$P_k = \begin{cases} \text{Probability of } |v_i^f| = 0, & k = 0 \\ \text{Probability of } 2^{k-1} \leq |v_i^f| < 2^k, & k \in \mathbb{N} \end{cases}$$

The exact evaluation of these probabilities is computationally prohibitive since it requires the entire image data to be processed. However, good estimates of the probability can be obtained by sampling the frames at a rate, $1 : R_{sample}$, that provides an appropriate balance between accuracy and computation. We compared the original probability distribution of the multiplier without sampling with the distribution achieved from various sample rates. We empirically find that $R_{sample} = 16$ provides a good match between the original and sampled distributions, while also providing a significant reduction in computation. The sampling framework is implemented by randomizing all N_f image frames and the probabilities are computed using first N_f/R_{sample} frames that act as representative of remaining frames. To obtain P_k probability values, we have designed a probability calculation block, which we will discuss shortly.

C. Dynamic Approximation Hardware

The dynamic approximate architecture for each multiplier node, m_i, consists of following three blocks:

1) the dynamic multiplier block
2) the probability calculation block
3) the optimization block

We will discuss about the hardware description of the dynamic multiplier block and the probability calculation block in this section. The optimization block will be discussed in Section VI.

The multiplier m_i is designed based on a CSA-based configuration, as mentioned in Section III, and we use only the lower triangle of this architecture. If the bit width of the multiplier is N_{m_i}, then it has N_{m_i} columns of FAs, and the x^{th} of these columns contains x FAs. The total number of FAs in the multiplier is:

$$\lambda_{m_i} = \sum_{x=0}^{N_{m_i}-1} x \tag{16}$$

In static approximation, the lower α_{m_i} columns use approximate FA blocks and rest of the columns employ accurate

TABLE II: A list of notations for various power terms

Notation	Explanation	Notation	Explanation
$\mathcal{P}^x_{a_i}$ $x = \{1, 2\}$	Power of an adder at node a_i, operating under the following modes $x = 1$: Accurate N_{a_i}-bit adder $x = 2$: Static approximate α_{a_i}-bit adder	$\Delta\mathcal{P}^x_{DCT}$ $x = \{2, 3\}$	Power savings for various approximation modes for 2D DCT block over all frames $x = 2$: Static approximation $x = 3$: Dynamic approximation
$\mathcal{P}^x_{m_i}$ $x = \{1, 2, 3\}$	Power of a multiplier at node m_i, operating under the following modes $x = 1$: Accurate N_{m_i}-bit multiplier $x = 2$: Static approximate α_{m_i}-bit multiplier $x = 3$: Dynamic approximate $(N_{m_i} - \alpha_{m_i})$-bit multiplier	$\Delta\mathcal{P}^x_{m_i}$ $x = \{2, 3\}$	Power savings for various approximation modes implementations at multiplier node m_i $x = 2$: Static approximation $x = 3$: Dynamic approximation
p_x $x = \{1, 2, 3, 4\}$	Power of an FA operating under the following modes $x = 1$: Accurate-mode FA $x = 3$: Accurate-mode DSFA $x = 2$: Approximate-mode FA $x = 4$: Approximate-mode DSFA	$\mathcal{P}_{abscomp}$	Power of the absolute comparator block
ϕ^f_i	Power of the DSFA in multiplier m_i for frame f	$\mathcal{P}_{opt}$	Power of the optimization block
$\Delta\mathcal{P}_{a_i}$	Total power savings for adder node, a_i	$\mathcal{P}_{prob}$	Power of probability calculation block
$\Delta\mathcal{P}_{m_i}$	Total power savings for multiplier node, m_i		

FAs. For dynamic approximation, the accurate FA columns are replaced with the columns of dynamic select FA (DSFA) blocks. An additional absolute comparator block is added to each dynamic multiplier to compare v^f_i and $2^{\mathcal{N}_i}$.

Based on sampled image-specific data obtained from the probability calculation block, the optimization block chooses an appropriate z_i. The absolute comparator block checks whether for the specific image, $|v^f_i| \leq 2^{\mathcal{N}_i+z_i}$. If so, it asserts the signal, $\mathcal{S}^f_i(z_i)$ and triggers dynamic full bit-width approximation. This full bit-width approximation increases the error variance to $\sigma^2_{m,true}(\alpha_{m_i} + z_i)$ as mentioned in (12). The formulation to obtain selector signal per frame, $\mathcal{S}^f_i(z_i)$, is

$$\mathcal{S}^f_i(z_i) = \begin{cases} 1, & \text{if } |v^f_i| < 2^{(\mathcal{N}_i+z_i)} \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

The capability of dynamically changing the level of approximation is enabled through the use of the DSFA block, shown in Fig. 5(a), consisting of an accurate FA, an approximate FA, and a MUX that chooses between the two, based on the selector signal, $\mathcal{S}^f_i(z_i)$ [15]. When the approximate FA is in operation, the accurate FA block is power-gated to reduce the power. An optimized DSFA design has been proposed, shown in Fig. 5(b), that skips the MUX delay of the accurate mode signals and thus reduces the critical delay as well as the total power of the DSFA design.

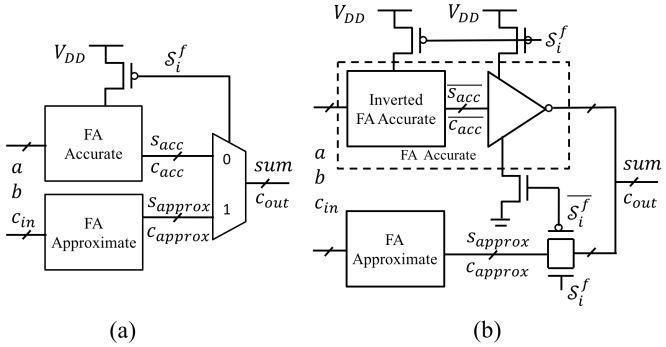


Fig. 5: (a) Preliminary and (b) optimized structure of a dynamic select FA (DSFA).

During dynamic approximation, the selector signal, $\mathcal{S}^f_i(z_i)$, decides between two operating modes: static approximation and full bit-width approximation. If $\lambda_{m_i,s}$ and $\lambda_{m_i,d}$ are the number of FAs selected for static and dynamic approximation alone, respectively, then for an α_{m_i}-bit approximation,

$$\lambda_{m_i,s} = \textstyle\sum_{x=0}^{\alpha_{m_i}-1} x \;\;,\;\; \lambda_{m_i,d} = \sum_{x=\alpha_{m_i}}^{N_{m_i}-1} x \tag{18}$$

For each N_{m_i}-bit dynamic multiplier node, m_i, we have designed a probability calculation block. The block has total N_{m_i} counters with the bitwidth $\log_2(N_f/R_{sample})$. Each k^{th} counter is enabled by a logic circuit that detects whether the input signal lies in the range $2^{k-1} \leq |v^f_i| < 2^k$.

D. Power Modeling

A list of variables used to model the power dissipation of various blocks under a set of static and dynamic approximation schemes is provided in Table II. For the accurate and statically approximated multiplier, the power expressions are:

$$\mathcal{P}^1_{m_i} = \textstyle\sum_f p_1 \cdot \lambda_{m_i} = \sum_f p_1[\lambda_{m_i,s} + \lambda_{m_i,d}] \tag{19}$$

$$\mathcal{P}^2_{m_i} = \textstyle\sum_f p_2 \cdot \lambda_{m_i,s} \tag{20}$$

To compute the multiplier power for the dynamic approximation case, we begin with modeling the DFSA. The DFSA operates in the accurate mode except when the selector signal, $\mathcal{S}^f_i(z_i)$, is asserted. The power dissipation in the f^{th} frame, $\phi^f_{m_i}(z_i)$, for the DSFA is given by:

$$\phi^f_{m_i} = \left(1 - \mathcal{S}^f_i(z_i)\right) \cdot p_3 + \mathcal{S}^f_i(z_i) \cdot p_4 \tag{21}$$

The larger the time $\mathcal{S}^f_i = 1$, the lower is the power for DSFA block, which depends on larger z_i, according to (17). The power for a dynamically approximated multiplier is:

$$\mathcal{P}^3_{m_i} = \textstyle\sum_f \left[\phi^f_{m_i}(z_i) \cdot \lambda_{m_i,d} + \mathcal{P}_{abscomp}\right] + \mathcal{P}_{overhead} \tag{22}$$

where $\mathcal{P}_{abscomp}$ is the power dissipation of the absolute comparator. The last term is the overhead of dynamic optimization:

$$\mathcal{P}_{overhead} = \left(\frac{N_f}{R_{sample}}\right) \cdot \mathcal{P}_{prob} + \mathcal{P}_{opt} \tag{23}$$

where $\mathcal{P}_{prob}$ and $\mathcal{P}_{opt}$ are, respectively, the power dissipation of the probability calculation block and the optimization block, discussed in Section VI. In (22) and (23), note that the dynamic multiplier block operates in all N_f frames of the image, the probability calculation block on (N_f/R_{sample}) frames, and the optimization block only once on the entire image.

We launch dynamic approximation from the static approximation case. The power savings over the accurate case are:

$$\Delta\mathcal{P}_{m_i} = \mathcal{P}^1_{m_i} - [\mathcal{P}^2_{m_i} + \mathcal{P}^3_{m_i}] = \Delta\mathcal{P}^2_{m_i} + \Delta\mathcal{P}^3_{m_i} \quad (24)$$

$$\text{where} \quad \Delta\mathcal{P}^2_{m_i} = \textstyle\sum_f \lambda_{m_i,s} \cdot (p_1 - p_2)$$

$$\Delta\mathcal{P}^3_{m_i} = \textstyle\sum_f \left[\lambda_{m_i,d} \cdot \left(p_1 - \phi^f_{m_i}(z_i)\right) - \mathcal{P}_{abscomp}\right] - \mathcal{P}_{overhead}$$

The terms $\Delta\mathcal{P}^2_{m_i}$ and $\Delta\mathcal{P}^3_{m_i}$ indicate, respectively, the power savings of static and dynamic approximation. The former is constant over all frames, but the latter varies with each frame.

The total dynamic power savings in the DCT block is the sum of power savings over all dynamic multiplier blocks:

$$\Delta\mathcal{P}^3_{DCT} = \textstyle\sum_{m_i \in \theta_m} \Delta\mathcal{P}^3_{m_i} \quad (25)$$

where θ_m is the set of all multipliers in the 2D DCT block.

VI. Optimized Dynamic Approximation

A. Formulation of the Optimization Problem

The goal of optimized dynamic approximation is to perform a set of inexpensive optimizations in real time. As a result, it is important for the optimizations to be simple. We achieve this by *locally* maximizing the power savings, $\Delta\mathcal{P}^3_{m_i}(z_i)$, at each node of the DCT operation. The optimization operates the dynamic multiplier nodes with the highest power savings subject to error constraints that guarantee output quality. At each multiplier node m_i, we solve the problem:

$$\begin{aligned} &\max_{z_i} \quad \Delta\mathcal{P}^3_{m_i}(z_i) \\ &\text{subject to} \quad \sigma^2_{m,true}(\alpha_{m_i} + z_i) \le \sigma^2_m(\alpha_{m_i}) \end{aligned} \quad (26)$$

From (24), the power savings, $\Delta\mathcal{P}^3_{m_i}(z_i)$, are maximized when $\phi^f_{m_i}(z_i)$ is as small as possible, and from (17) and (21), this occurs when z_i is maximized. However, the value of z_i is limited by the error variance budget, $\sigma^2_m(\alpha_{m_i})$. Here, the true error variance, $\sigma^2_{m,true}(\alpha_{m_i} + z_i)$, is defined by (15). Therefore, we simplify the optimization problem to:

$$\max \quad z_i, \quad \text{s.t.} \quad \sigma^2_{m_i,true}(\alpha_{m_i} + z_i) \le \sigma^2_{m_i}(\alpha_{m_i}) \quad (27)$$

B. Design Space Reduction for the Optimization Problem

1) A Motivating Example: A key question relates to the choice of the error budget, $\sigma^2_m(\alpha_{m_i})$, at each multiplier in (27). One way to set the budget is to choose the value from static optimization, allowing each node to be dynamically configurable, but this requires substantial overhead circuitry at each node. Instead, we limit configurability to nodes with the largest potential for power savings, and increase their error budgets to utilize the unused error budget from the other nodes.

To understand the relation between error at multiple nodes in a DAG, consider the example DAG where two multiplier nodes, A and B, converge at an exact adder node D. We define the error slack variance, $\sigma^2_{m_i,slack}$, as:

$$\sigma^2_{m_i,slack} = \sigma^2_m(\alpha_{m_i}) - \sigma^2_{m,true}(\alpha_{m_i}) \quad (28)$$

$$= (1 - \eta_{m_i})\,\sigma^2_m(\alpha_{m_i}) \quad (29)$$

where $\eta_{m_i} = \sigma^2_{m,true}(\alpha_{m_i}) / \sigma^2_m(\alpha_{m_i})$. The error slack variance at nodes A and B from the static approximation budget

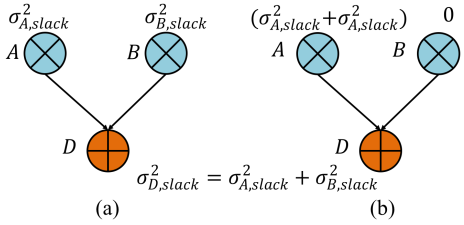


Fig. 6: The output error variance slack (a) without and (b) with the benefit of error variance stacking.

(Fig. 6(a)) could be insufficient to allow an extra approximate bit at either node. However, the total error variance at the output is the sum of the error variances at all nodes in the DAG, and we could stack the error variances by allocating the entire slack budget to node A, as in Fig. 6(b), i.e., A would be the only dynamically configurable node, and the error slack of B would be transferred to A. This has following benefits:

- It may allow A to further approximate one or more bits, saving power while remaining within the slack budget.
- The hardware overhead incurred of dynamic reconfigurability is halved if only A is made reconfigurable, since B can be approximated statically without this overhead.

2) A Framework for Redistributing Slack Budgets: We have implemented this slack redistribution idea for the DAG model of the 1D DCT structure shown in Fig. 2. There are 11 multiplier nodes, m_1 through m_{11}, and eight outputs, x_0 through x_7, in the 1D DCT structure. The candidates for dynamic approximation are all the multiplier nodes. We denote the sensitivity of the error at node x_j for the error at m_i as S_{ij} and the corresponding sensitivity of the variance of the error as S^2_{ij}. For each (multiplier, output) pair, (m_i, x_j), of the DCT structure, the error variance sensitivity, S^2_{ij}, is shown in Table III. A blank entry at (m_i, x_j) implies no connection between m_i and x_j. Outputs x_0 and x_4 have no multipliers in their fan-in cone and are not shown. The three distinct sensitivity values, s_1, s_2 and s_3, are listed below the table.

TABLE III: Variance sensitivities for multipliers in 1D DCT block

$S^2_{i,j}$	x_1	x_2	x_3	x_5	x_6	x_7
m_1	s_1		s_2			s_1
m_2	s_1			s_2		s_1
m_3	s_1		s_2			s_1
m_4	s_1			s_2		s_1
m_5	$2s_1$		s_2	s_2		$2s_1$
m_6	$2s_1$		s_2	s_2		$2s_1$
m_7					s_1	
m_8		s_1				
m_9		s_1			s_1	
m_{10}			s_3			
m_{11}				s_3		

$s_1 = \frac{1}{2^{14}}, s_2 = \frac{181^2}{2^{28}}$ and $s_3 = \frac{1}{2^{28}}$

Let us use $\mathcal{M}_j$ to denote the set of multiplier nodes that are on a path to output x_j The total error variance introduced

by the static approximation process at x_j is:

$$\begin{aligned}\sigma^2_{x_j,stat} &= \textstyle\sum_{i\in\mathcal{M}_j} S^2_{i,j}\cdot\sigma^2_m(\alpha_{m_i}) \\ &= \textstyle\sum_{i\in\mathcal{M}_j} S^2_{i,j}\cdot\left[\sigma^2_{m,true}(\alpha_{m_i})+\sigma^2_{m_i,slack}\right] \quad (30)\end{aligned}$$

where the second equality follows from (28). As indicated in the example above, we may use more approximation and also save on the overhead of dynamic reconfiguration by making only a few nodes dynamically reconfigurable and stacking the slacks of the other nodes on to these nodes. We therefore partition the elements of $\mathcal{M}_j$ into a set of nodes that are selected for dynamically approximation, $\mathcal{M}_{j,d}$, and a set of nodes, $\mathcal{M}_{j,s}$, that are left at their static approximation levels.

As we redistribute the slacks of the static nodes to the dynamic nodes, we associate each dynamic node, $m_i \in \mathcal{M}_{j,d}$, with a set of static nodes, $\mathcal{Y}_{m_i}$. As we stack the slack from the static nodes to the dynamic nodes, the variance at output x_j becomes:

$$\begin{aligned}\sigma^2_{x_j,dyn} &= \sum_{i\in\mathcal{M}_{j,d}}\left[S^2_{i,j}\cdot\sigma^2_m(\alpha_{m_i})+\sum_{m_k\in\mathcal{Y}_{m_i}} S^2_{k,j}\cdot\sigma^2_{m_k,slack}\right] \\ &\quad+\sum_{i\in\mathcal{M}_{j,s}} S^2_{i,j}\cdot\sigma^2_{m,true}(\alpha_{m_i}) \\ &= \sum_{i\in\mathcal{M}_{j,d}} S^2_{i,j}\cdot\left[\sigma^2_m(\alpha_{m_i})+\sum_{m_k\in\mathcal{Y}_{m_i}}\chi_{m_i}[m_k]\cdot\sigma^2_{m_k,slack}\right] \\ &\quad+\sum_{i\in\mathcal{M}_{j,s}} S^2_{i,j}\cdot\sigma^2_{m,true}(\alpha_{m_i}) \quad (31)\end{aligned}$$

where the weights $\chi_{m_i}[m_k] = S^2_{k,j}/S^2_{i,j}$.

The above equation provides a new error budget, $\sigma^2_{m_i,NB}$, for nodes $m_i \in \mathcal{M}_{j,d}$, which can be simplified using the approach provided in Appendix A:

$$\sigma^2_{m_i,NB} = \tilde{\mathcal{C}}_{m_i} - \tilde{\mathcal{D}}_{m_i}\times\sigma^2_{m,true}(\alpha_{m_i}) \quad (32)$$

Here, $\tilde{\mathcal{C}}_{m_i}$ and $\tilde{\mathcal{D}}_{m_i}$ are error budget dependent constants for node, m_i. The new budget, $\sigma^2_{m_i,NB}$, is used to update the optimization problem from (27) to:

$$\max \quad z_i, \quad \text{s.t.} \quad \sigma^2_{m,true}(\alpha_{m_i}+z_i) \le \sigma^2_{m_i,NB} \quad (33)$$

We propose two following algorithms to reduce the design space of the optimization problem:

- A heuristic described in Algorithm 1 selects a set of dynamically approximated nodes, Λ.
- For each multiplier $m_i \in \Lambda$, Algorithm 2 identifies the set of non-dynamic nodes, $\mathcal{Y}_{m_i}$, whose slacks are stacked on to dynamic node m_i, and the set of weights, χ_{m_i}, associated with stacking, as described in (43) of Appendix A.

3) An Algorithm for Selecting Dynamic Multiplier Nodes: The criteria for selecting a set of dynamic nodes ensure that:

1) the chosen nodes cover all outputs, i.e., the propagated errors reach all outputs (with the exception of x_0 and x_4, which cannot be reached by any multiplier).
2) a goodness metric, ω_{PB}, explained in Appendix B, ensures high approximation levels with low error, is minimized.
3) the error at the outputs of the 1D DCT block is within the total error budget, thus ensuring that dynamic approximation saves power while staying within error specifications.

Algorithm 1 Selection of Dynamic Multiplier Nodes, Λ

INPUT: Multiplier nodes in the 1D DCT network, $\{m_1,\ m_2,\cdots,\ m_{11}\}$ and the corresponding dynamic error variance sensitivities: $\{\omega_1,\ \omega_2,\ \cdots,\ \omega_{11}\}$.
OUTPUT: A set, Λ, of dynamically approximated nodes.

1: $\Lambda = \emptyset$
2: $S_m = \{m_1,\ m_2,\cdots,\ m_{11}\}$
3: **if** $m_i, m_j \in S_m, i \neq j$ have identical rows in Table III **then**
4: $\quad S_m = S_m \setminus m_j$
5: **end if**
6: Sort S_m, arranging m_is in non-ascending order of ω_i.
7: **for** each multiplier node $m_i \in S_m$ **do**
8: $\quad$ **if** Discarding m_i loses control on any output **then**
9: $\quad\quad \Lambda = \Lambda \cup m_i$
10: $\quad$ **end if**
11: **end for**
12: $\omega_{min} \leftarrow \infty$
13: **for** ν = $|S_m|$ to 1 **do**
14: $\quad$ Calculate $\omega_{PB} = \left(\sum_{k=1}^{\nu}\omega_{S_m(k)}\right)/\nu$
15: $\quad$ **if** $\omega_{PB} < (1+\epsilon)\times\omega_{min}$ **then**
16: $\quad\quad \omega_{min} = \omega_{PB}$, $\Lambda = \Lambda \cup S_m(\nu)$
17: $\quad$ **end if**
18: **end for**
19: **return**

The inputs to Algorithm 1 are all the multiplier nodes in the 1D DCT network and their corresponding dynamic error variance sensitivities defined in Appendix B. If two nodes have identical row entries in Table III, we discard one of them from consideration for Λ, as shown in Line 3–5 of Algorithm 1. For example, since the rows corresponding to m_1 and m_3 nodes are identical, m_1 could be pruned out. The rationale for this is based on the large overhead of creating a DSFA to allow reconfigurability: even though the error increases exponentially with z_i, for real test cases, the break-even point where the benefits of approximating more multipliers under an error budget overcomes this overhead is well beyond the range of the eventually selected values of z_i.

Line 6 sorts the elements m_i of set S_m in non-ascending order of ω_i. Next, in Lines 7–11, we identify any nodes that can uniquely influence some output x_j, i.e., if these nodes are not included in Λ, x_j cannot be reached. Based on Criterion 1) above, such nodes are added to Λ. After this step, we greedily select from the remaining nodes in non-decreasing order of ω_i. In Lines 13–17, we add a node to Λ if it reduces ω_{PB}, using a factor of ϵ to allow temporary cost increases that permit a level of hill-climbing during optimization. The dominating operation of this algorithm is to check the influence of each candidate multiplier node to all the output nodes. Hence, the complexity of this algorithm is $O(\mathcal{T}_x\cdot\mathcal{T}_m)$. Here, $\mathcal{T}_x$ and $\mathcal{T}_m$ are the total number of output and multiplier nodes in a DAG, respectively.

4) An Algorithm for Redistributing Error Slacks: Algorithm 1 yields $\Lambda = \{m_3, m_4, m_7, m_8, m_{10}, m_{11}\}$, i.e., 6 of 11 multiplier nodes are chosen for dynamic approximation.

Further practical considerations are used to prune the number of candidate multiplier nodes over the entire JPEG hardware. As shown in Fig. 1(b), there are two layers of 1D DCT blocks in the 2D DCT, and the constant multiplications in the second layer amplify any error introduced by the first layer of 1D DCT block. This implies that there is limited advantage in implementing dynamic approximation for the multiplier nodes of Layer 1, and therefore, we only choose multiplier nodes from Layer 2 for dynamic approximation. In summary, we choose a total of $6 \times 8 = 48$ multiplier nodes, listed in a set, θ''_m, for dynamic approximation out of $11 \times 16 = 176$ available multiplier nodes.

Algorithm 2 Algorithm to obtain $\mathcal{Y}_{m_i}$ and χ_{m_i}

INPUT: Λ = Set of selected multiplier nodes,
$\mathcal{M}_{j,d}$ = Set of nodes $m_i \in \Lambda$ that have a path to output x_j
$\mathcal{M}_{j,s}$ = Set of nodes $m_i \notin \Lambda$ with a path to output x_j
$\mathcal{O}_{m_i}$ = Set of outputs x such that $m_i \in \Lambda$ has a path to x
OUTPUT: $\mathcal{Y}_{m_i}$= Nodes whose error slacks are stacked to $m_i \in \Lambda$
$\chi_{m_i} = \text{map}\left\langle \text{key} = m_k, \text{value} = \frac{S^2_{k,j}}{S^2_{i,j}} \right\rangle, m_k \in \mathcal{Y}_{m_i}, m_i \in \Lambda$

1: Initialize $\mathcal{Y}_{m_i} \leftarrow \varnothing, \chi_{m_i} \leftarrow \varnothing, V \leftarrow \varnothing \ \ \forall m_i \in \Lambda$
2: Create $\mathcal{I}_{m_i} = \bigcap_{x_j \in \mathcal{O}_{m_i}} \mathcal{M}_{j,s}, \quad \forall m_i \in \Lambda$
3: **for** each $x_j \quad j \leftarrow \{0, 1, \cdots, \mathcal{T}_x - 1\}$ **do**
4: Calculate $\sigma^2_{x_j,dyn}$ using (31)
5: **if** $\sigma^2_{x_j,dyn} > \sigma^2_{x_j,stat}$ **then**
6: $\chi_{m_i}[m_k] = \min\left(\chi_{m_i}[m_k], \frac{S^2_{k,j}}{S^2_{i,j}}\right),$
7: $\forall m_k \in \mathcal{Y}_{m_i}, m_i \in \mathcal{M}_{j,d} \cap V$
8: **end if**
9: $\mathcal{M} = \mathcal{M}_{j,d} \setminus V$
10: $\mathcal{I}_{m_i} = \mathcal{I}_{m_i} \setminus \cup_{m_i \in \mathcal{M}} \mathcal{Y}_{m_i}, \quad \forall m_i \in \mathcal{M}$
11: Calculate $\mathcal{Q}_j = \bigcap_{m_i \in \mathcal{M}} \mathcal{I}_{m_i}$
12: Assign $\mathcal{Y}_{m_i} \leftarrow \mathcal{I}_{m_i} \setminus \mathcal{Q}_j, \quad \forall m_i \in \mathcal{M}$
13: $Y[m_i] = \mathcal{Y}_{m_i}, \quad \forall m_i \in \mathcal{M}$
14: $Y = \text{NodeDist}(Y, \mathcal{Q}_j, \mathcal{M})$
15: $\mathcal{Y}_{m_i} = Y[m_i], \quad \forall m_i \in \mathcal{M}$
16: $\chi_{m_i}[m_k] = S^2_{k,j}/S^2_{i,j}, \forall m_k \in \mathcal{Y}_{m_i}, \forall m_i \in \mathcal{M}$
17: $V \leftarrow V \cup \mathcal{M}$
18: **end for**
19: **return**

The inputs to the Algorithm 2, $\mathcal{M}_{j,d}$, $\mathcal{M}_{j,s}$, and $\mathcal{O}_{m_i}$, and the dynamic node set, Λ, can be obtained from Table III and Algorithm 1. After the initialization in Line 1, Line 2 creates $\mathcal{I}_{m_i}$, the set of nodes that are candidates for stacking on m_i: these are nodes that lie on every output path, $\mathcal{O}_{m_i}$, that m_i can reach. The rationale for this choice can be illustrated by an example: node $m_3 \in \Lambda$ reaches $\mathcal{O}_{m_3} = \{x_1, x_3, x_7\}$, and the non-dynamic nodes that lie on every path to nodes in this set are $\mathcal{I}_{m_3} = \{m_1, m_5, m_6\}$. Hence, stacking the error slacks of m_1, m_5, or m_6 to m_3 is merely a redistribution of slacks that will not result any error constraint violation. Now consider node $m_2 \notin \mathcal{I}_{m_3}$, which lies on the path to x_1 and x_7, but not x_3. Therefore, stacking the error slacks of m_2 to m_3 will add a new error on the path to x_3, potentially violating the error budget at x_3. Hence, m_2 is not a stacking candidate for m_3.

Algorithm 3 Y =NodeDist$(Y, \mathcal{Q}_j, \mathcal{M})$

1: **if** $\mathcal{Q}_j \neq \varnothing$ **then**
2: Find $\sigma^2_{x_j,m_k}$, $m_k \in (\cup_{m_i \in \mathcal{M}} Y[m_i]) \cup \mathcal{Q}_j$, using (45)
3: $TS[m_i] = \sum_{m_k \in Y[m_i]} \sigma^2_{x_j,m_k}, \forall m_i \in \mathcal{M}$
4: Sort $\mathcal{Q}_j$ in descending order of $\sigma^2_{x_j,m_k}, m_k \in \mathcal{Q}_j$
5: **for** $m_k \in \mathcal{Q}_j$ **do**
6: $m_l \leftarrow$ argmin(TS)
7: $Y[m_l] \leftarrow Y[m_l] \cup \{m_k\}$
8: $TS[m_l]$ += $\sigma^2_{x_j,m_k}$
9: **end for**
10: **end if**
11: **return** Y

Next, we consider paths to each output x_j in an iterative loop. The present value of the dynamic budget for output x_j, $\sigma^2_{x_j,dyn}$, is first computed in Line 4. If this value exceeds the variance budget from static optimization, then the slacks are adjusted to ensure adherence to the budget by updating the χ_{m_i} values of the visited node set V in accordance with (43).

We then consider the set $\mathcal{M}$ of dynamic nodes that have not been visited so far on Line 9. The nodes in the set $\mathcal{Y}_{m_i}$ are removed from further stacking consideration on Line 10 as they have already been stacked to m_i. These nodes can no longer share their slacks, and we update a narrower set of candidates, $\mathcal{I}_{m_i}$. We then compute the set $\mathcal{Q}_j$ in Line 11, which contains non-dynamic nodes on every path from every element of $\mathcal{M}$ to an output. This set, $\mathcal{Q}_j$, helps to identify the unique terms in $\mathcal{I}_{m_i}$ that are directly assigned to $\mathcal{Y}_{m_i}$ in Line 12. For example, for output x_1, $\mathcal{M}_{1,s} = \{m_3, m_4\}$, and $\mathcal{I}_{m_3} = \{m_1, m_5, m_6\}$, $\mathcal{I}_{m_4} = \{m_2, m_5, m_6\}$, so that $\mathcal{Q}_1 = \{m_5, m_6\}$. The unique elements of $\mathcal{I}_{m_3}$ and $\mathcal{I}_{m_4}$, m_1 and m_2, are directly assigned to $\mathcal{Y}_{m_3}$ and $\mathcal{Y}_{m_4}$, respectively.

On Line 13, we assemble Y, a set of all sets $\mathcal{Y}_{m_i}$, which is updated in the function call to NodeDist on Line 14, where the slacks of the nodes in $\mathcal{Q}_j$ are distributed among $Y[m_i]$ (i.e., $\mathcal{Y}_{m_i}$). After the function returns the updated value of Y, the values are written back into the $\mathcal{Y}_{m_i}$ variables. Finally, Line 16 updates the list χ_{m_i} for nodes $m_i \in \mathcal{M}$, and Line 17 marks the nodes in $\mathcal{M}$ as visited.

The NodeDist routine is described in Algorithm 3. It first computes the slacks of all nodes belongs to the Y and $\mathcal{Q}_j$ sets on Line 2. The total slacks of the stacked nodes in $Y[m_i]$ for all $m_i \in \mathcal{M}$ are calculated in Line 3. The sorted nodes in $\mathcal{Q}_j$ are then greedily stacked to the dynamic nodes in Lines 5–9. The complexity of this algorithm is $O(|\mathcal{M}| \cdot |\mathcal{Q}_j|)$. Since the cardinality of both $\mathcal{M}$ and $\mathcal{Q}_j$ can be in the order of total number of multiplier nodes, $\mathcal{T}_m$, of a DAG, the complexity becomes: $O\left(\mathcal{T}_m^2\right)$. The complexity of Algorithm 2 is dominated by the operation of NodeDist routine for all outputs, $\mathcal{T}_x$. Hence, the overall complexity of this algorithm is $O\left(\mathcal{T}_x \cdot \mathcal{T}_m^2\right)$.

The results of Algorithm 2 are shown in Table IV, which lists the stacked nodes, $m_k \in \mathcal{Y}_{m_i}$, and the corresponding weights, $\chi_{m_i}[m_k]$, for each $m_i \in \Lambda$. The blank entries of $\mathcal{Y}_{m_i}$ and χ_{m_i} in the table imply that no stacking is possible

for the corresponding node, and the budget, $\sigma^2_{m_i,NB}$, for such a node is unchanged from its static budget, $\sigma^2_m(\alpha_{m_i})$.

TABLE IV: The set of stacked nodes, $\mathcal{Y}_{m_i}$, and the corresponding weights χ_{m_i} for all selected nodes $m_i \in \Lambda$

$m_i \in \Lambda$	m_3	m_4	m_7	m_8	m_{10}	m_{11}
$m_k \in \mathcal{Y}_{m_i}$	m_1, m_5	m_2, m_6	m_9	m_9	–	–
$\chi_{m_i}[m_k]$	1, 2	1, 2	1	1	–	–

C. Real-Time Optimization of Dynamic Approximation

The optimization problem (33) must be solved in real time to obtain the result of dynamic approximation level, z_i. To control the area and power overhead associated with dynamic approximation, it is essential to simplify the framework described above, addressing both hardware and runtime overheads.

1) Reducing the Number of Iterations during Optimization: The number of dynamic approximate bits, z_i, is a non-negative integer valued number and could be increased in steps of one. The use of a coarser resolution, $\kappa > 1$, can reduce the number of iterations, potentially at the cost of optimality. Practically, $\kappa = 2$ provides a good balance between the two.

2) Approximating Algorithm and Hardware: We introduce several heuristics to reduce the computation.

a) Approximation of the Error Variance Formula: The computations in the optimization problem involve several additions and multiplications, which would consume significant power if implemented as precise floating point additions and multiplications. To mitigate this, we have approximated the multiplier error variance equation (2) as:

$$\hat{\sigma}^2_m(\alpha) = 0.5 \times 4^{\alpha} = 2^{(2\alpha-1)} \tag{34}$$

This approximation converts the multiplication operation to a shift, thus saving a notable amount of power. We show in Section VII-B2 that under this approximation, we obtain significant area and power savings at the system level.

b) Approximation of the Hardware: As stated in Section V-C, an absolute comparator is required in the dynamic multiplier block. The calculation of the absolute value of an n-bit number requires n XOR gates that XOR the sign bit with the data bits, and an n-bit adder that adds the sign bit to the result to obtain the two's complement of a negative number. To reduce the hardware cost, we omit the adder block: as a result, the absolute value of a negative number is off by one, which causes no significant error in our estimation.

c) Formulating the Iterations Incrementally: In the optimization problem (33), the left hand side of the constraint is given by (15) and involves the computation of $\gamma_1(z_i)$ and $\gamma_2(z_i)$, The optimization procedure iteratively increments the values of some z_is, and we devise recursive expressions that update these values between iterations incrementally as:

$$\gamma_1(z_i) = \gamma_1(z_i - \kappa) + \mathcal{F}_1(\kappa, z_i) \tag{35}$$

$$\gamma_2(z_i) = 4^{\kappa} \cdot [\gamma_2(z_i - \kappa) - \mathcal{F}_2(\kappa, z_i)] \tag{36}$$

$$\mathcal{F}_1(\kappa, z_i) = \textstyle\sum_{v=\mathcal{N}_i+z_i-\kappa}^{\mathcal{N}_i+z_i-1} P_v \cdot \hat{\sigma}^2_m(v + \lceil \log_2(c_i) \rceil) \tag{37}$$

$$\mathcal{F}_2(\kappa, z_i) = \hat{\sigma}^2_m(\alpha_{m_i} + z_i - \kappa) \cdot \textstyle\sum_{v=\mathcal{N}_i+z_i-\kappa}^{\mathcal{N}_i+z_i-1} P_v \tag{38}$$

where the base cases, $\gamma_1(0)$ and $\gamma_2(0)$ are obtained from (15). Note that the updated formula uses the simplification in (34).

The operations for updating γ_1 and γ_2 are implemented in hardware as a combinational block. From (34), the $\hat{\sigma}^2_m$ term is a power of 2, and therefore $\mathcal{F}_1$ and hence γ_1 can be implemented using adders and shifters, without expensive multiplications. Similarly, $\mathcal{F}_2$ and γ_2 can also be implemented using adders and shifters, and therefore the hardware overhead associated with implementing these updates is low.

The overall dynamic multiplier architecture for multiplier node, m_i, is shown in Fig. 7. For an input image, the probability values, P_k, are calculated in the probability calculation block using reduced number of frames. Using these P_k values, the optimization block first calculates the base case values of γ_1 and γ_2 using (15) and later computes the new budget, $\sigma^2_{m_i,NB}$ from (32). For each z_i, in steps of κ, the error variance is calculated using (35)-(38) and compared against the new budget, $\sigma^2_{m_i,NB}$, in parallel. The encoder block detects the maximum feasible value of z_i and passes the value to the dynamic multiplier block.

The percentage power savings for the DCT block under static and dynamic approximation condition incorporating all the heuristics is given by:

$$\begin{aligned} \frac{\Delta\mathcal{P}_{DCT}}{\mathcal{P}_{DCT}} &= \frac{\Delta\mathcal{P}^2_{DCT} + \Delta\mathcal{P}^3_{DCT}}{\sum_{m_i\in\theta_m}\mathcal{P}^1_{m_i} + \sum_{a_i\in\theta_a}\mathcal{P}^1_{a_i}} \\ \text{where } \Delta\mathcal{P}^2_{DCT} &= \textstyle\sum_{m_i\in\theta_m}\mathcal{P}^2_{m_i} + \sum_{a_i\in\theta_a}\Delta\mathcal{P}_{a_i} \\ \Delta\mathcal{P}^3_{DCT} &= \textstyle\sum_{m_i\in\theta''_m}\mathcal{P}^3_{m_i} \\ \mathcal{P}^1_{a_i} &= \textstyle\sum_f N_{a_i} \times p_1,\ \Delta\mathcal{P}_{a_i} = \sum_f \alpha_{a_i} \times (p_1 - p_2) \end{aligned} \tag{39}$$

where θ_a and θ_m are the sets of all adders and multipliers, respectively, and θ''_m is the set of all dynamic nodes in the 2D DCT block. Using (19) and (24), the power savings of static approximation can be reduced to

$$\frac{\Delta\mathcal{P}^2_{DCT}}{\mathcal{P}_{DCT}} = \left(\frac{p_1-p_2}{p_1}\right) \cdot \left[\frac{\sum_{m_i\in\theta_m}\lambda_{m_i,s} + \sum_{a_i\in\theta_a}\alpha_{a_i}}{\sum_{m_i\in\theta_m}\lambda_{m_i} + \sum_{a_i\in\theta_a}N_{a_i}}\right] \tag{40}$$

The first term, $(\frac{p_1-p_2}{p_1})$, is the power savings for the approximate FA w.r.t. the accurate FA, and the second term is the number of approximate FAs compared to the total number of FAs in the 2D DCT block. The power savings for static approximation is constant for all images for a specified δ_R. The required area, power, and delay of the dynamic approximation process including the overheads are studied in details in Section VII.

VII. Results

We have implemented our approximate DCT framework using the 45nm NanGate Open Cell Library. The absolute comparator, probability calculation, and optimization block have each been synthesized from an RTL description using Synopsys Design Vision to obtain the power, delay, and area associated with these blocks. The FA blocks (accurate, approximate, DSFA) were designed using Cadence Virtuoso and simulated with HSPICE to obtain their power and delay, and the areas were estimated by drawing the layouts of these FA blocks. All simulations have been performed at the typical process corner with $V_{DD} = 1.1$V and temperature, T = 25°C. Table V lists the

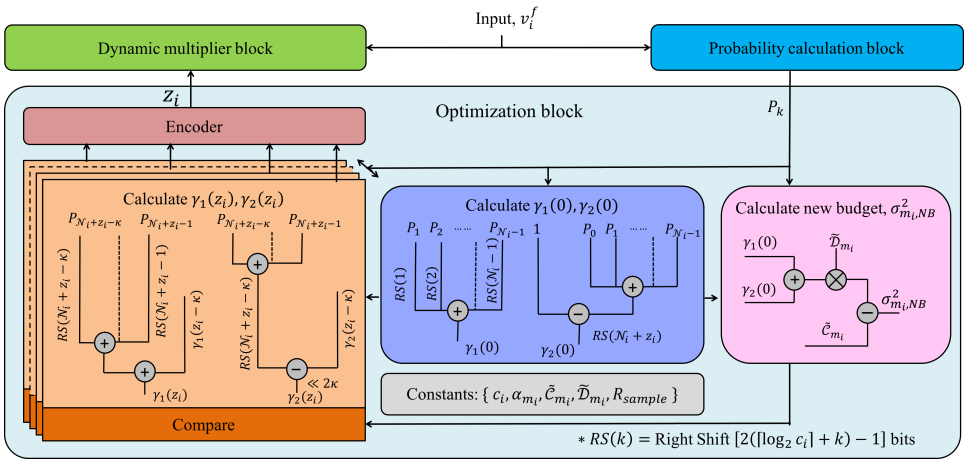


Fig. 7: Dynamic multiplier architecture for node, m_i.

power requirements for the building blocks. The JPEG circuit was exercised on the benchmark images from [17], [18].

A. Static optimization

We first present the results of static optimization, employing the mixed integer nonlinear problem solver, KNITRO [19], to solve the static nonlinear optimization problem (11). The CPU times for optimization were around 1 hour on a 2.6 GHz Intel Core i5 CPU with 8Gb RAM and running the 64-bit OS X. This computation time is reasonable since this is a one-time optimization. The optimized values of α_{a_i} and α_{m_i} obtained from KNITRO were used in (40) to calculate total power savings using the statically optimized approximate DCT hardware. The area of the circuitry was obtained by synthesizing the blocks using the design flow described above.

The first two columns of Table VI show power savings from static approximation for various values of user-specified maximum pixel error, δ_R (defined in (7)), at node, R, of Fig. 1(a). As expected, the magnitude of savings increases when the error specification is relaxed. It is also worth noting that the magnitude of power savings is image-independent, motivating our work on dynamic approximation.

B. Dynamic optimization

Next, we consider the case of dynamic approximation, where the 48 multipliers identified in Section VI-B4 may be dynamicaly approximated beyond their static approximation solution, in an image-specific manner. We have used the granularity level, $\kappa = 2$ with all proposed heuristics described in Section VI-C for this work. The probability distributions at the multiplier inputs, the switching rates of various blocks as well as the static approximate solution were used in (39) to find the total static and dynamic power savings.

1) Power savings for a set of input images: For five representative images, Columns 3–7 of Table VI show the overall power savings, incorporating the cost of the overhead circuitry. The last column shows a theoretical limit on the best achievable power savings, corresponding to the case where all bits of the 48 dynamic multiplier nodes are approximated. The Gray21 image achieves this limit in all cases, and the other images require a certain number of precise bits and deliver power savings that are below the limit.

The dynamic approximation level variations for an error budget, $\delta_R = 30$, are shown in Fig. 8 for image (a) Lena and (b) Baboon. Each plot depicts the level of introduced error at the 64 outputs under static and dynamic approximation. We analyze the results for each image below.

Lena: Moving from static to dynamic approximation increases the error but keeps it within specifications.

Baboon: Static approximation alone brings the error to the constrained limit, and no further gains are achieved through dynamic approximation. In fact, as shown in Table VI, the power gains from static approximation are somewhat lost due to the overhead of the dynamic approximation circuitry. At higher error specifications (e.g., $\delta_R = 60$), dynamic approximation provides gains over static approximation.

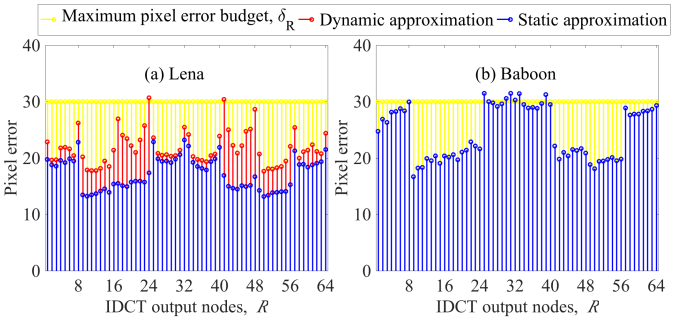


Fig. 8: The pixel error at each output for (a) Lena and (b) Baboon ($\delta_R = 30, F = 90$).

The dynamic power savings varies with different images which can be attributed to various pixel contrasts of the images. The pixel contrast can be defined as the difference in pixel values among the adjacent image coordinates. As shown in Fig. 2, the inputs to the multipliers of 1D DCT block come from subtractor outputs, and we focus on differences of pairwise pixel inputs to stage, $s(1)$ of 1D DCT for each column of matrix, M, as defined in Section II. We take the cumulative distribution function, CDF, of 10 bins of the pixel

TABLE V: Power dissipation of various blocks

Blocks	Delay (ns)	Power (μW)
Accurate FA	0.16	5.49
Approximate FA	0.00	0.00
DSFA (Accurate)	0.20	6.94
DSFA (Approximate)	0.08	1.13
Absolute (XOR) comparator	1.28	49.09
Probability calculation block	0.10	129.04
Optimization block	6.15	5.6×10^3

TABLE VI: Power savings for a set of input images

Max. pixel error, δ_R	%Power savings						
	Static	Dynamic					Theoretical limit
		Baboon	Peppers	Lena	Arctichare	Gray21	
30	34.8	32.5	38.0	46.0	46.0	50.3	50.3
40	38.8	37.5	45.7	49.5	50.5	53.5	53.5
50	41.2	41.6	49.0	52.8	52.9	55.8	55.8
60	43.3	45.3	52.7	55.8	55.1	57.2	57.2
70	44.7	47.4	55.4	57.4	56.8	58.2	58.2
80	45.8	49.4	56.7	58.6	58.2	59.3	59.3
90	46.8	51.0	57.8	58.7	58.7	59.8	59.8

difference values, and define the contrast for an image as the index of the bin which contains the 95 percentile data. For example, the image contrast level for Arctichare image is 2 as the second bin contains the 95 percentile data as shown in Fig. 9(a). On the other hand, Baboon image has larger contrast level (shown in Fig. 9(b)) than Arctichare as more bins are required to reach 95 percentile data.

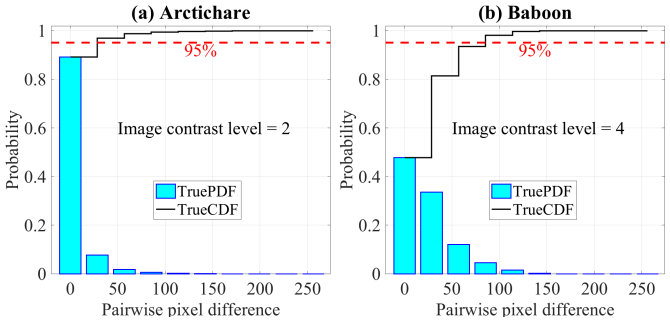


Fig. 9: Distribution of pairwise pixel differences and image contrast level calculation of (a) Arctichare and (b) Baboon.

If M^f and R^f are the f^{th} frame of the original and reconstructed image matrices, respectively, as shown in Fig. 1, and N_f is the total number of frames for the image, we define the mean square error, $MSE = \frac{1}{N_f}\sum_f \left(M^f - R^f\right)^2$. We quantify the quality degradation of an image using the peak signal-to-noise ratio ($PSNR$) metric, defined as:

$$PSNR = 10\log\left(\frac{255^2}{MSE}\right) \tag{41}$$

We use MATLAB to compute the image $PSNR$ values.

The heatmaps in Fig. 10 show average of percentage dynamic power savings, $\Delta\mathcal{P}^3_{DCT}/\mathcal{P}_{DCT}$, and average of image quality degradation from the base case, ΔPSNR, for 30 standard images [17], [18] with different contrast levels and error budgets, δ_R. The base case corresponds to a quality factor, $F = 90$ with no approximation, and the $PSNR$ values are computed using (41). For $\delta_R = 0$, the dynamic approximation process was implemented without any static approximation. Hence, the power savings are negative due to the overhead of additional circuitry. For the same image contrast level, as the level of static approximation increases, the overhead becomes progressively easier to overcome which improves the dynamic power savings but degrades PSNR.

On the other hand, increased image contrast level translates to higher probabilities of higher input ranges to multipliers and introduces larger error as described in (15). As a result, for the same error budget, increased image contrast degrades both dynamic power savings and PSNR. We can create a lookup table using this heatmap. For a known maximum image contrast level of a particular application, the required minimum error budget for dynamic approximation to outperform static approximation can be found. If the average PSNR degradation for the minimum error budget is acceptable for the target application, then the JPEG hardware can be designed with dynamic approximation capabilities.

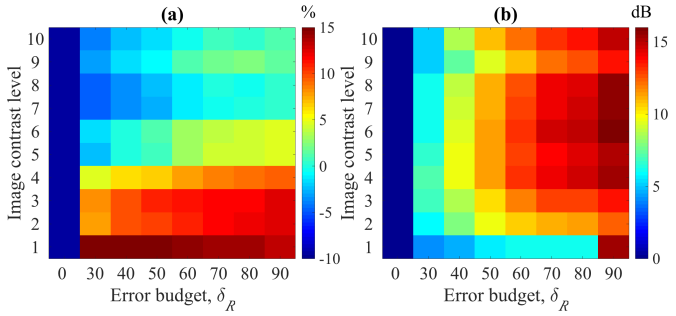


Fig. 10: Heatmap of average (a) percentage dynamic power savings, $\Delta\mathcal{P}^3_{DCT}/\mathcal{P}_{DCT}$, and (b) $PSNR$ degradation for various image contrast levels and error budgets, δ_R.

2) Impact of hardware choices: We evaluate the effectiveness of our heuristics in Section VI-C that simplify the hardware implementations of the absolute comparator, the error variance formula in (34), and the choice of the granularity level, κ, of the number of dynamic approximate bits, z_i in Table VII. For several choices of κ and under various heuristics, we compare the power and area savings and the $PSNR$. We evaluate these on the Baboon image, which shows the lowest power savings, i.e., the highest overhead, in Table VI. Comparing the second row with the first, for the same value of κ, the introduction of heuristics show significant area savings and some power savings, with negligible change in the $PSNR$. The use of coarser granularity of z_i, shown in the third row, cuts into the power savings and $PSNR$ slightly, but uses noticeably less area. due to the reduction in the optimization circuitry.

3) Impact of approximation choices: To quantify the gains of specific approaches, we focus on the image Gray21, which is seen to achieve the maximum theoretical savings. We compare the following three approaches to achieve iso-quality degradation ($\delta_R = 30$) for a specific image, Gray21:

TABLE VII: Comparison of power and area savings, and PSNR for various hardware choices (Baboon and $\delta_R = 70$)

Hardware choice		% Power savings	% Area overhead	$PSNR$ (dB)
Heuristics	κ			
No	1	46.69%	27.96%	22.01
Yes	1	47.81%	5.42%	22.02
Yes	2	47.41%	1.73%	22.12

1) No approximation with a small quality factor, $F = 50$
2) Static approximation with a large quality factor, $F = 90$
3) Dynamic approximation with $F = 90$

We assume the base case to be the no-approximation JPEG computation with the quality factor, $F = 90$. For a maximum pixel error, $\delta_R = 30$, the compression ratio, power, delay and area comparison for various approaches, with respect to the base case, are listed in Table VIII. The compression ratio is inversely related to F, and is calculated from the image storage sizes of the original image and JPEG compressed image. The base case uses a $9:1$ compression ratio.

TABLE VIII: Comparison of memory requirement, power, delay, area savings and normalized power-delay product w.r. to dynamic approximation (Gray and $\delta_R = 30$)

	Reduced Quality	Static approximation	Dynamic approximation	
			(a)	(b)
Quality factor, F	50	90	90	90
Relative compression ratio	15:9	1:1	1:1	1:1
% power savings	0.0%	34.8%	50.1%	50.3%
% delay savings	0.0%	46.1%	41.1%	41.0%
% area savings	0.0%	34.8%	–91.3%	–12.2%
Normalized power-delay product	3.4×	1.2×	1.0×	1.0×

Approach 1) uses a small quality factor, $F = 50$ to achieve the target δ_R. The storage requirement is reduced by $(15/9)\times$ compared to the baseline. However, this approach does not yield any area, power, and delay savings as there is no approximation. On the other hand, Approach 2) employs static approximation with $F = 90$ and reduces the power and area by 34.8% and delay by 46.1% over the base case.

Approach 3) uses $F = 90$, as in Approach 2), but implements dynamic approximation. A simple implementation of this scheme requires a large amount of overhead circuitry, and therefore we explore tradeoffs that will reduce this overhead. We consider the structure of the dynamic multiplier node, as shown in Fig. 7. The optimization block has the largest area overhead, and we consider two choices:

(a) Each dynamic multiplier node is provided with its own optimization block.
(b) The dynamic aproximation was implemented in each of the eight 1D DCT blocks in Layer 2 of the 2D DCT block of Fig. 1(b). Since the multiplier nodes of these eight blocks use the same constant multiplier values and the same static approximation levels, we use a single optimization block for all 1D DCT blocks.

The results for both cases are shown in the last two columns of Table VIII. Case (a) incurs a large area overhead of over 91% over the base case, but Case (b) reduces this area requirement significantly to 12%. The optimization block computes the dynamic approximation level, z_i, for all the dynamic multipliers. For case (a), all calculations are performed in parallel and requires only one additional clock cycle, but for case (b), the shared optimization block finds the optimal z_i values for each of the eight blocks serially and thus requires eight additional clock cycles. This incurs a small delay penalty, but this additional overhead is negligible in comparison with the delay and power of the DCT block computations. The power gain for this approach is 50.3% over the baseline, which translates to $\left(\frac{65.2-49.7}{65.2}\right) \times 100\% = 23.8\%$ lower power requirement than the static approximation alone. However, dynamic approximation process requires additional overhead circuitry and hence the delay requirement for this approach is increased by $\left(\frac{59.0-53.9}{53.9}\right) \times 100\% = 9.5\%$ than the static approximation. As the power improvement is more dominating than the delay increment, the static approximation process requires $\left(\frac{65.2\times 53.9}{49.7\times 59.0}\right) = 1.2\times$ larger power-delay product compared to the dynamic approximation process. Whereas, the power-delay product for approach 1) is significantly larger than both static and dynamic approximation process.

VIII. Conclusion

We have proposed an optimized JPEG compression unit framework for a user-specified error budget, consisting of both static design-time optimization and dynamic run-time approximation. Depending on the input image characteristics, the approximation level changes to meet the user specified error budget, translating into overall power savings.

Appendix A
Formulation of New Error Budget

A new error budget, $\sigma^2_{m_i,NB}$, for nodes $m_i \in \mathcal{M}_{j,d}$, obtained from (31) is combined with (29) to get:

$$\sigma^2_{m_i,NB} = \sigma^2_m(\alpha_{m_i}) + \sum_{m_k \in \mathcal{Y}_{m_i}} \chi_{m_i}[m_k] \cdot (1 - \eta_{m_k}) \cdot \sigma^2_m(\alpha_{m_k}) \tag{42}$$

The value of $\sigma^2_{m_i,NB}$ can be different for each output x_j. Since the error variance constraint at each output must be satisfied, $\sigma^2_{m_i,NB}$ must be chosen in such a way that $\sigma^2_{x_j,dyn} \leq \sigma^2_{x_j,stat}, \forall x_j$. In other words, the value of $\chi_{m_i}[m_k]$ must be chosen so that the redistributed slack on a path to any output x_j does not exceed $\sigma^2_{x_j,stat}$. This is achieved by setting

$$\chi_{m_i}[m_k] = \min_{\text{all outputs } j} \left[S^2_{k,j}/S^2_{i,j}\right] \tag{43}$$

In addition, the computation of (42) can be further simplified. In practice, for all $m_k \in \mathcal{Y}_{m_i}$, η_{m_k} have very similar values. Approximating each such term as η_{m_i}, we have:

$$\begin{aligned}\sigma^2_{m_i,NB} &\approx \sigma^2_m(\alpha_{m_i}) + (1 - \eta_{m_i}) \times \sum_{m_k \in \mathcal{Y}_{m_i}} \chi_{m_i}[m_k] \cdot \sigma^2_m(\alpha_{m_k}) \\ &= \tilde{\mathcal{C}}_{m_i} - \tilde{\mathcal{D}}_{m_i} \times \sigma^2_{m,true}(\alpha_{m_i}) \end{aligned} \tag{44}$$

where $$\tilde{\mathcal{C}}_{m_i} = \sigma^2_m(\alpha_{m_i}) + \sum_{m_k \in \mathcal{Y}_{m_i}} \chi_{m_i}[m_k] \cdot \sigma^2_m(\alpha_{m_k}),$$

$$\tilde{\mathcal{D}}_{m_i} = \sum_{m_k \in \mathcal{Y}_{m_i}} \chi_{m_i}[m_k] \cdot \sigma^2_m(\alpha_{m_k})/\sigma^2_m(\alpha_{m_i})$$

The evaluation of (44) requires the computation of $\sigma^2_{m,true}(\alpha_{m_i})$, $\tilde{\mathcal{C}}_{m_i}$, and $\tilde{\mathcal{D}}_{m_i}$. The true error variance, $\sigma^2_{m,true}(\alpha_{m_i})$, is image-dependent and is obtained using a sampling procedure using (15). On the other hand, both $\tilde{C}_{m_i}$ and $\tilde{\mathcal{D}}_{m_i}$ are constant for node m_i for a specific error budget, and are determined using $\sigma^2_m(\alpha_{m_i})$ (obtained from (2) using the α_{m_i} value from the solution of the static optimization problem, (11)), $\mathcal{Y}_{m_i}$, and χ_{m_i}.

Appendix B
A Goodness Metric for Dynamic Node Selection

As a stepping stone to defining ω_{PB}, we introduce a total error variance metric, $\sigma^2_{m_i,T}$, for multiplier m_i. If a dynamic node m_i uses z_i more approximate bits beyond static approximation, the error variance at output x_j is:

$$\sigma^2_{x_j,m_i} = S^2_{ij} \times \sigma^2_m(\alpha_{m_i} + z_i) \tag{45}$$

The total error variance over all eight outputs due to m_i is:

$$\sigma^2_{m_i,T} = \sum_{j=0}^{7} \sigma^2_{x_j,m_i} = \sigma^2_m(\alpha_{m_i} + z_i) \sum_{j=0}^{7} S^2_{ij} \tag{46}$$

The dynamic error variance sensitivity, ω_i, is the sensitivity of $\sigma^2_{m_i,T}$ to z_i for a unit increase in approximation as:

$$\omega_i = \left. \frac{d\sigma^2_{m_i,T}}{dz_i} \right|_{z_i=1} = \ln(4) \times \sigma^2_m(\alpha_i + 1) \times \sum_{j=0}^{7} S^2_{ij} \tag{47}$$

where the last expression follows from (2) and (46). To determine whether a node is a good candidate for dynamic reconfiguration, its value of ω_i should be sufficiently small.

We define per-bit error, ω_{PB}, as the average value of ω_i over the elements of Λ. This serves as a goodness metric for selecting dynamic nodes for inclusion in Λ.

Acknowledgment

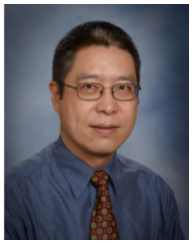
This work was supported in part by the NSF under awards CCF-1162267, CCF-1525925, and CCF-1525749.

References

[1] V. Gupta, *et al.*, "Low-Power Digital Signal Processing Using Approximate Adders," *IEEE T. Comput. Aid. D.*, vol. 32, no. 1, pp. 124–137, 2013.
[2] G. Varatkar *et al.*, "Energy-Efficient Motion Estimation using Error-Tolerance," in *Proc. ISLPED*, pp. 113–118, 2006.
[3] Y. Emre *et al.*, "Energy and Quality-Aware Multimedia Signal Processing," *IEEE T. Multimedia*, vol. 15, no. 7, pp. 1579–1593, 2013.
[4] L. N. Chakrapani, *et al.*, "Highly Energy and Performance Efficient Embedded Computing Through Approximately Correct Arithmetic: A Mathematical Foundation and Preliminary Experimental Validation," in *Proc. CASES*, pp. 187–196, 2008.
[5] W. Chen, *et al.*, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE T. Commun.*, vol. 25, no. 9, pp. 1004–1009, 1977.
[6] B. Lee, "A New Algorithm to Compute the Discrete Cosine Transform," *IEEE T. Acoust. Speech*, vol. 32, no. 6, pp. 1243–12457, 1984.
[7] C. Loeffler, *et al.*, "Practical Fast 1-D DCT Algorithms with 11 Multiplications," in *Proc. ICASSP*, vol. 2, pp. 988–991, 1989.
[8] A. Mammeri, *et al.*, "Modeling and Adapting JPEG to the Energy Requirements of VSN," in *Proc. ICCCN*, pp. 1–6, 2008.
[9] J. Park, *et al.*, "Dynamic Bit-Width Adaptation in DCT: An Approach to Trade Off Image Quality and Computation Energy," *IEEE T. VLSI Syst*, vol. 18, no. 5, pp. 787–793, 2010.
[10] K. Nepal, *et al.*, "ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits," in *Proc. DATE*, pp. 361:1–361:6, 2014.
[11] C. Li, *et al.*, "Joint Precision Optimization and High Level Synthesis for Approximate Computing," in *Proc. DAC*, pp. 1–6, 2015.
[12] F. S. Snigdha, *et al.*, "Optimal Design of JPEG Hardware Under the Approximate Computing Paradigm," in *Proc. DAC*, pp. 1–6, 2016.
[13] D. Sengupta, *et al.*, "SABER: Selection of Approximate Bits for the Design of Error Tolerant Circuits," in *Proc. DAC*, 2017.
[14] Z.-L. He, *et al.*, "Low-Power VLSI Design for Motion Estimation Using Adaptive Pixel Truncation," *IEEE T. Circuits Syst. Video Technol.*, vol. 10, no. 5, pp. 669–678, 2000.
[15] A. Raha, *et al.*, "Input-Based Dynamic Reconfiguration of Approximate Arithmetic Units for Video Encoding," *IEEE T. VLSI Syst*, vol. 24, no. 3, pp. 846–857, 2016.
[16] V. Bhaskaran *et al.*, *Image and Video Compression Standards: Algorithms and Architectures*. Kluwer Academic Publishers, 1997.
[17] "Standard Image Database." http://sipi.usc.edu/database/. Accessed February 22, 2018.
[18] N. Asuni *et al.*, "TESTIMAGES: A Large Data Archive For Display and Algorithm Testing," *Journal of Graphics Tools*, vol. 17, no. 4, pp. 113–125, 2013.
[19] "KNITRO User Manual." https://www.artelys.com/tools/knitro_doc/. Accessed February 22, 2018.

Farhana S. Snigdha received the B.Sc. degree from Bangladesh University of Engineering and Technology, Dhaka and is currently pursuing Ph.D. degree in Electrical Engineering at the University of Minnesota. She has interned at Cadence Design Systems, Austin, TX, USA. Her research interests include design and optimization of low power system and architecture, and is currently working on real-time approximate low-power image processing as well as various Neural Network architectures.

Deepashree Sengupta received the B. Tech. and M.Tech degrees from the Indian Institute of Technology, Kharagpur, and the PhD degree from the University of Minnesota. She has interned at Taiwan Semiconductor Manufacturing Company Limited and Intel Corporation, Santa Clara, USA. Her research interests include circuit reliability issues in nanometer scale regime, and low power design of CMOS circuits. She is a recipient of the Doctoral Dissertation Fellowship from the University of Minnesota, and is currently a Senior R&D Engineer at Synopsys Inc.



Jiang Hu (F'16) received the B.S. degree from Zhejiang University (China), the M.S. and the Ph.D. degrees from the University of Minnesota. He has worked with IBM Microelectronics from 2001 to 2002, and has been a faculty at the Texas A&M University, thereafter. His research interests include VLSI circuit optimization, adaptive design, hardware security, and power management for large scale computing systems. He has received two conference Best Paper awards, the IBM Invention Achievement Award, and the Humboldt Research Fellowship. He was an associate editor of IEEE TCAD from 2006 to 2011, and is currently an associate editor for the ACM Transactions on Design Automation of Electronic Systems.

Sachin S. Sapatnekar (S'86, M'93, F'03) received the B. Tech. degree from the Indian Institute of Technology, Bombay, the M.S. degree from Syracuse University, and the Ph.D. degree from the University of Illinois. He taught at Iowa State University from 1992 to 1997 and has been at the University of Minnesota since 1997, where he holds the Distinguished McKnight University Professorship and the Robert and Marjorie Henle Chair in the Department of Electrical and Computer Engineering. He has received seven conference Best Paper awards, a Best Poster Award, two ICCAD 10-year Retrospective Most Influential Paper Awards, the SRC Technical Excellence award and the SIA University Researcher Award. He is a Fellow of the ACM and the IEEE.