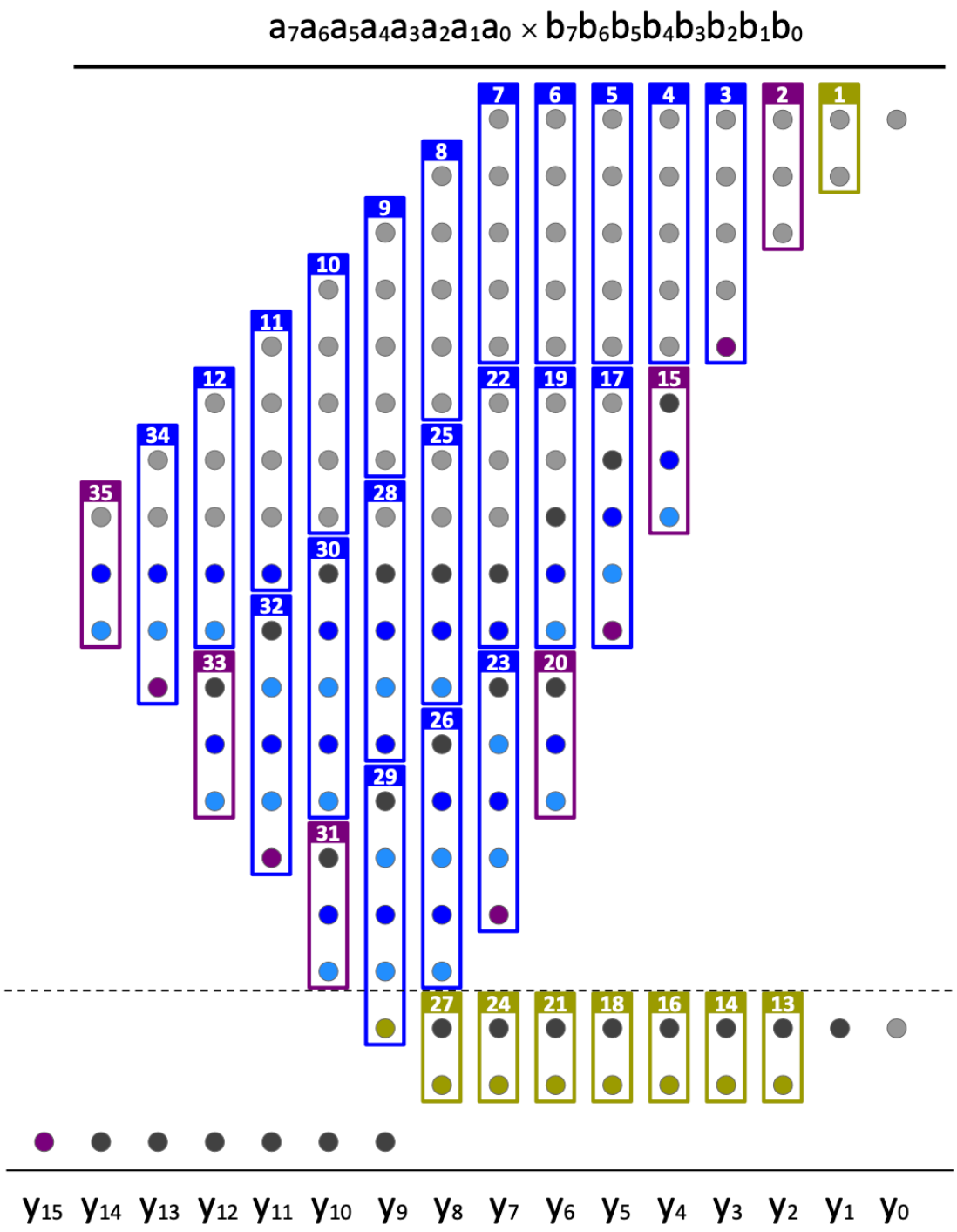
1. Introduction

In digital signal processing, image processing, and deep learning inference, a large number of arithmetic operations (especially multiplication) often account for most of a system's power consumption and computational delay. To balance performance, power, and area requirements, **Approximate Computing** has become an increasingly important research direction. Its core concept is that, as long as a certain degree of computational error is acceptable, one can simplify the hardware circuitry or computational process to gain higher energy efficiency or lower latency.

This proposal is based on an **8×8-bit unsigned integer multiplier**. We present three different approximate design approaches for this multiplier. We also compare their performance in image filtering, image enhancement, and deep neural network inference, assessing various metrics such as error distance, image quality (PSNR, SSIM), and classification accuracy.

---

2. Baseline Model



The figure above shows a **standard 8×8-bit unsigned integer multiplier**. Its computation typically involves three stages:

1. **Partial Product Generation**
2. **Partial Product Compression and Addition**
3. **Final Summation**

In the hardware implementation, the following basic units are often used:

- **FA (Full Adder)**
- **HA (Half Adder)**
- **CA (4-2 Compressor)**

An exact multiplier strictly follows each unit's truth table, ensuring that the output perfectly matches the ideal multiplication result.

---

3. Approximation Methods

To reduce hardware complexity, latency, and power consumption, we propose three approximation approaches tailored to the baseline multiplier described above. These approaches achieve various trade-offs between accuracy and hardware savings, primarily by **omitting or altering the carry signals** in certain addition or compression units.

3.1 Approximation Method 1: Selective Approximation (`approximation_1`)

In this strategy, we leverage a **Bayesian optimization** process that chooses which FAs, HAs, or CAs to approximate based on a specified objective function (e.g., hardware cost, output error). The main operations are:

- **For FAs (Full Adders) and HAs (Half Adders)**
  - **ignore_cout**: Ignore the adder's carry-out (i.e., fix the output carry to 0).
- **For CAs (4-2 Compressors)**
  - **ignore_cout**: Ignore the compressor's carry-out.
  - **ignore_carry**: Ignore the compressor's carry-in.
  - **ignore_cout_and_carry**: Ignore both the carry-in and carry-out signals.

Below is an example configuration where specific units (e.g., `fa2`, `fa20`, `ca17`) are approximated according to different rules:

```
approximation_1 = {
    "FA": {
        "ignore_cout": ["fa2", "fa20"]
    },
    "HA": {
        "ignore_cout": ["ha1", "ha13", "ha14", "ha21"]
    },
    "CA": {
        "ignore_cout_and_carry": ["ca17", "ca28", "ca6", "ca9"],
        "ignore_carry": ["ca25", "ca29", "ca3", "ca32", "ca8"],
        "ignore_cout": ["ca26", "ca30", "ca4", "ca5"]
    }
}
```

3.2 Approximation Method 2: Replacing All CA Units with a Custom LUT (`lut_4_2`)

This more radical approach replaces **all** 4-2 compressors (CA) in the multiplier with a single custom lookup table, `lut_4_2`. While the inputs remain (a, b, c, d, cin) and outputs remain (sum, carry, cout), the LUT's internal truth table does not strictly match the exact 4-2 compression logic and is instead designed to simplify or omit certain carries. An excerpt of the LUT looks like this:

```
lut_4_2 = {
    (0,0,0,0,0): (0, 0, 1),
    (0,0,0,0,1): (0, 1, 0),
    (0,0,0,1,0): (0, 1, 0),
    ...
    (1,1,1,1,1): (1, 1, 1)
}
```

By globally applying the same approximate LUT to every CA, one obtains a more uniform and potentially simpler hardware implementation, but at the cost of higher output error in many cases.

3.3 Approximation Method 3: Hybrid Approach (`approximation_3`)

The third approach combines elements of the first two but **only applies approximations to CA units** (4-2 compressors), leaving all FA and HA units exact. For each CA, one of the following four strategies is chosen:

1. Use the same LUT as in Method 2 (`lut`).
2. Ignore the carry-out (`ignore_cout`).
3. Ignore the carry-in (`ignore_carry`).
4. Ignore both the carry-in and carry-out (`ignore_cout_and_carry`).

Using Bayesian optimization again, one might end up with a mixed configuration such as:

```
approximation_3 = {
    "CA": {
        "lut": ["ca3", "ca4", "ca5", "ca7", "ca17", "ca23"],
        "ignore_cout": ["ca6", "ca26", "ca12", "ca19", "ca22", "ca26", "ca29", "ca30",
"ca34"],
        "ignore_carry": ["ca9", "ca25", "ca32"],
        "ignore_cout_and_carry": ["ca8"]
    }
}
```

---

4. Performance and Error Evaluation

To thoroughly assess each approximate approach, we employ various metrics:

1. **Error (E)** and **Error Distance (ED)**
    - The absolute difference between the approximate result and the exact result.
2. **MRED (Mean Relative Error Distance)**
    - The average of the error distance normalized by the true result.
3. **PSNR (Peak Signal-to-Noise Ratio)** and **SSIM (Structural Similarity Index Measure)**

    - Used to evaluate image quality in tasks such as Gaussian filtering, edge detection, and image sharpening.
4. **Classification Accuracy**
    - Used in deep neural network inference (e.g., ResNet18, DenseNet121 on ImageNet or CIFAR10) to compare the final classification accuracy when the exact multipliers are replaced by the approximate multipliers.

4.1 Approximation Method 1 (`approximation_1`)

- **Error (E)**:

```
count    65535.000000
mean       986.327550
std       1781.237389
min          0.000000
25%         32.000000
50%        448.000000
75%       1280.000000
max      11580.000000
```

- **Mean Relative Error Distance (MRED)**:

```
1.9122935955649842e-19
```

4.2 Approximation Method 2 (`lut_4_2`)

- **Error (E)**:

```
count    65535.000000
mean     14966.310521
std       3106.561459
min       1144.000000
25%      13256.000000
50%      15800.000000
75%      17040.000000
max      20984.000000
```

- **Mean Relative Error Distance (MRED)**:

```
1.0763481095037197e-16
```

4.3 Approximation Method 3 (`approximation_3`)

- **Error (E)**:

```
count    65535.000000
mean       813.021317
std       1029.327689
min          0.000000
25%        184.000000
50%        352.000000
75%       1120.000000
max       6376.000000
```

- **Mean Relative Error Distance (MRED)**:

```
1.1473761573389906e-18
```

From these measurements, we observe that:

- `approximation_2` (the global LUT replacement) shows the largest average and relative errors overall.
- `approximation_1` and `approximation_3` exhibit significantly lower error, especially `approximation_3`, which can preserve good accuracy while requiring relatively modest hardware modifications.

---

5. Image Processing and Classification Tasks

Below is a complete comparison table in English for various image processing tasks (e.g., blending, edge detection, sharpening), showing SSIM (Structural Similarity Index Measure) and PSNR (Peak Signal-to-Noise Ratio in dB) for the outputs generated by each approximate approach versus the exact multiplier.

Exact Image	Approximate Image	SSIM	PSNR (dB)
exact_blend1.png	appro1_blend1.png	0.8498	24.19
exact_blend1.png	appro2_blend1.png	0.5593	17.42
exact_blend1.png	appro3_blend1.png	0.8141	24.67
exact_blend2.png	appro1_blend2.png	0.9266	24.05
exact_blend2.png	appro2_blend2.png	0.4348	16.28
exact_blend2.png	appro3_blend2.png	0.8481	23.01
exact_blend3.png	appro1_blend3.png	0.8901	25.39
exact_blend3.png	appro2_blend3.png	0.5117	17.35
exact_blend3.png	appro3_blend3.png	0.8661	26.29
exact_edge_detection_astronaut.png	appro1_edge_detection_astronaut.png	1.0000	inf
exact_edge_detection_astronaut.png	appro2_edge_detection_astronaut.png	0.2083	6.81
exact_edge_detection_astronaut.png	appro3_edge_detection_astronaut.png	0.2128	6.85
exact_edge_detection_coffee.png	appro1_edge_detection_coffee.png	1.0000	inf
exact_edge_detection_coffee.png	appro2_edge_detection_coffee.png	0.2157	8.52
exact_edge_detection_coffee.png	appro3_edge_detection_coffee.png	0.2209	8.56
exact_edge_detection_rocket.png	appro1_edge_detection_rocket.png	1.0000	inf
exact_edge_detection_rocket.png	appro2_edge_detection_rocket.png	0.0322	6.01
exact_edge_detection_rocket.png	appro3_edge_detection_rocket.png	0.0333	6.02
exact_sharpened_astronaut.png	appro1_sharpened_astronaut.png	0.5769	12.15
exact_sharpened_astronaut.png	appro2_sharpened_astronaut.png	-0.0659	2.62
exact_sharpened_astronaut.png	appro3_sharpened_astronaut.png	-0.0315	2.77
exact_sharpened_coffee.png	appro1_sharpened_coffee.png	0.5470	11.25
exact_sharpened_coffee.png	appro2_sharpened_coffee.png	0.0046	4.33
exact_sharpened_coffee.png	appro3_sharpened_coffee.png	0.0448	4.49
exact_sharpened_rocket.png	appro1_sharpened_rocket.png	0.4781	11.29
exact_sharpened_rocket.png	appro2_sharpened_rocket.png	0.1848	2.30
exact_sharpened_rocket.png	appro3_sharpened_rocket.png	0.2034	2.40

From these results, we can see:

1. **Approximation Method 2** (`approximation_2`) often yields more severe image distortion, manifested in significantly reduced SSIM and PSNR.
2. **Approximation Methods 1** (`approximation_1`) and **3** (`approximation_3`) generally cause less distortion, and in some cases (e.g., blend1.png, blend3.png) their SSIM and PSNR values are close to those of the exact multiplier.
3. For tasks that are sensitive to local pixel differences (e.g., edge detection), **approximation_1** and **approximation_3** can still achieve decent performance.

In terms of **deep neural network inference**, experiments with ResNet18, DenseNet121 on ImageNet or CIFAR10 show that replacing exact multipliers with `approximation_1` or `approximation_3` only causes slight accuracy degradation. However, `approximation_2` leads to a more significant drop in classification accuracy. **NOT FINISHED!**

---

6. Conclusion and Future Work