# DATA-TRANSFORMATION ALGORITHMS

# FOR MINIMIZING BIT FLIPS ON GPU DATA BUSES

by

Arthi Nagarajan

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
August 2017

Committee Members:

Martin Burtscher, Chair

Apan Qasem

Zillang Zong

## FAIR USE AND AUTHOR'S PERMISSION STATEMENT

### Fair Use



### Duplication Permission

## *DEDICATION*

To my mentor, thank you for all of your support and guidance along the way. To my husband, thank you for your love and encouragement throughout my graduate program. To my parents, thank you for your never-ending love and support.

## ACKNOWLEDGEMENTS

This project is supported by the National Science Foundation and hardware donations from NVIDIA Corporation.

## TABLE OF CONTENTS

**Page**

## LIST OF TABLES

**Table** **Page**

## LIST OF FIGURES

**Figure** **Page**

## ABSTRACT

Excessive power consumption and cooling costs in computing centers as well as limited battery life in mobile devices make energy optimization an important area of research. In CMOS technology, dynamic energy is primarily consumed when switching from one state to another. In particular, the charging and discharging of long wires consume a significant amount of energy. GPU-based accelerators are widely used to solve many complex data-intensive problems, which tend to transfer large amounts of data to/from main memory. Therefore, GPU data buses contribute a significant portion of the total energy expenditure. As a consequence, encoding the data to minimize bit flips has the potential to greatly reduce the amount of energy consumed by data buses. The existing commercially available solution to reduce bit flips is called Data Bus Inversion (DBI). This thesis introduces more effective bit-flip minimization algorithms, which can eliminate about 9% more bit flips than DBI

# 1. INTRODUCTION

High reliability and low energy consumption was a major driving force behind the development of CMOS technologies. Power consumption of CMOS circuits can be classified into static power and dynamic power. Static power is always consumed when the power is on, irrespective of any activity, and dynamic power is consumed when the circuits are actively switching. I only target dynamic power in this thesis.

## 1.1 Dynamic power consumption of CMOS

The dynamic power consumption of a CMOS circuit is due to the current that flows through the device when transistors are switching from one logic state to another. The dynamic power consumption is given by

$$P_{dynamic} = C_L \bullet V^2 \bullet N \bullet f$$

Here, $P_{dynamic}$ represents the dynamic power, $C_L$ the capacitance, V the voltage, N the switching activity, and f the frequency [5].

The rate of switching has a direct impact on the current flow through the transistors and the power is dissipated due to the flow of that current, which is why $P_{dynamic}$ is proportional to both N and f. So, the higher the number of bit flips, the more energy is consumed. This thesis focuses on reducing such switching activity by encoding data on buses to improve energy efficiency, in particular of GPU memory buses.

## 1.2 Bit flips on data buses

In most computing devices, buses are used to transfer data from one component to another. The transfer of data over buses can consume a significant amount of energy due to the high capacitance of the long wires. A bit flip on a bus is a transition between one logic state and another. A significant amount of power is consumed in data buses due to

charging and discharging of long off-chip data bus wires, such as the ones that connect the GPU to its memory. GPU-based accelerators are used to solve data-intensive problems, many of which require lots of data transfers over the data bus. Thus, minimizing the bit flips by encoding data on these buses has the potential to significantly reduce the energy consumption of GPUs.

### 1.3 Data Bus Inversion

The existing solution to minimize the number of bit flips on a data bus is called Data Bus Inversion (DBI). DBI evaluates parallel data bits on the bus, and, depending on the flips between the current and previous data on the bus, a decision is made to invert or not invert all of the bits of the current data prior to transmission. A DBI bit is sent in parallel with the data over a dedicated bus line to specify whether the data has been inverted.

Figure 1 shows a Data Bus Inversion example. T1 through T4 represent time intervals. D0 through D3 represent the bits of the data to be transmitted. The data bus bits are assumed to be 0 before transmission. The bits highlighted in red are the bit flips, i.e., a transition from 0 to 1 or 1 to 0. The last row highlighted in grey is the dedicated extra wire for DBI. The overall number of bit flips in this example is reduced from 11 to 8, where 11 and 8 are the total number of red entries.

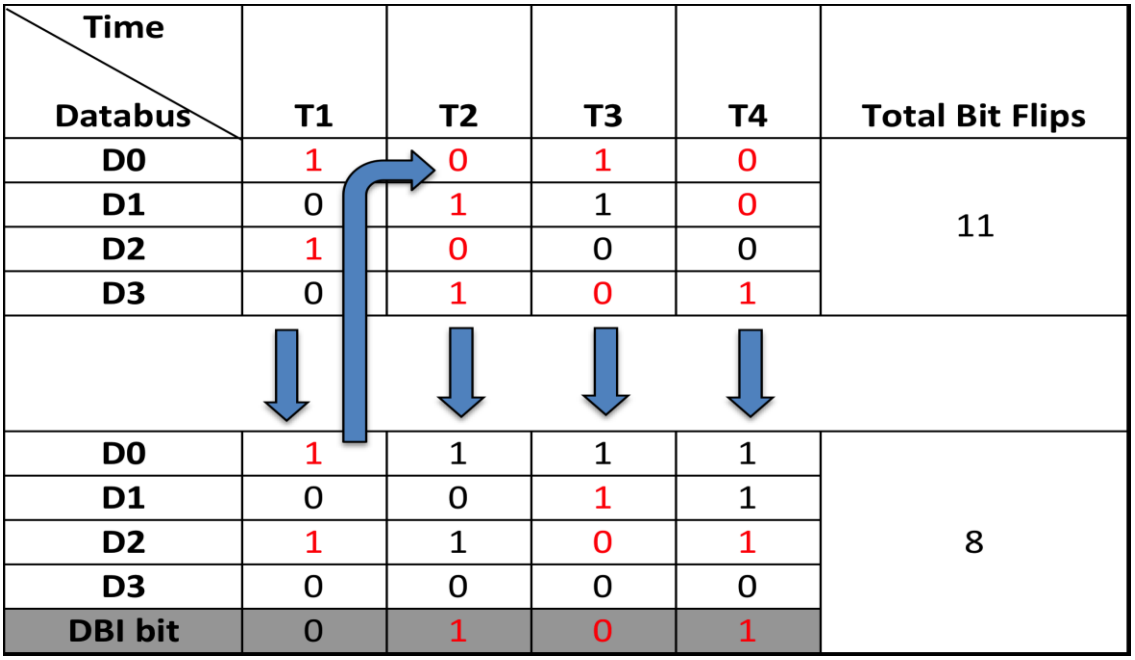


**Figure 1: Data Bus Inversion Example**

The DBI reduces the maximum number of transitions from n to n/2 where "n" is the bus width, thereby reducing the peak dynamic power consumption by half. This thesis includes a detailed study of DBI for different inputs and comparing it against my new solution. The reduction of bit flips using DBI for all inputs is ~22%. This thesis proposes a better bit flip minimization algorithm using a new approach called ECL-BFM, which can reduce bit flips up to ~20% on all input types.

### 1.4 ECL-BFM Algorithms

Data Bus Inversion is a fixed solution for all input types. This has obvious drawbacks and may be of help only for certain inputs. Moreover, DBI performs a very simple transformation on data, which is not necessarily effective for all input types.

ECL-BFM improves upon DBI by applying additional transformations on the input data before DBI, thereby reducing more bit flips than DBI does by itself. The additional transformations on the input data are built from simple algorithmic

components taken from various encoding algorithms. ECL-BFM approach is to link multiple such components together to generate an efficient algorithm for reducing the number of bit flips on various input data types. The reverse operations are performed to decode the data. DBI requires adding an extra bit to the hardware, which is very expensive. Hence, I also propose a solution that does not require any extra bit lines.

### 1.5 Contributions

This thesis makes the following contributions.

1. It proposes chaining of individual algorithmic components to generate effective algorithms to reduce the number of bit flips on data bus. The inverse of the individual algorithmic components can be applied in reverse order to decode the data.
2. It proposes to improve DBI by adding additional transformations as a preprocessing step to DBI. This approach can effectively reduce the number of bit flips for most inputs.
3. It proposes a low-cost bit-flip minimization algorithm that does not require additional bit lines.

### 1.6 Outline

The rest of the thesis is organized as follows: Chapter 2 presents the related work, Chapter 3 explains the design and implementation of the proposed approaches, Chapter 4 presents the evaluation methods, Chapter 5 explains the results and Chapter 6 concludes with summary and future work.

## 2. RELATED WORK

CMOS is a low-power technology, which is why it is widely used in computer systems. As explained in Section 1.1, $P_{dynamic}$ represents the dynamic power and is directly proportional to the switching activity N. Therefore, the energy consumption of a GPU, for example, can be reduced by reducing N. Low-power CMOS design are of particular interest to portable devices [7]. In this thesis, the "data" value is the data that has to be transmitted on the bus and the actual data on the bus is referred to as the "bus" value [7].

Low-power coding method called Bus Invert Coding or Data Bus Inversion (DBI) [7] was developed to reduce the switching activity on data buses. The DBI method requires one extra control bit called "invert" bit [7]. If the invert bit is zero, the bus value will be equal to the data value. If the invert bit is one, bus value will be the inverted data value. The number of bit flips can be decreased by DBI as follows:

- Compute the Hamming distance between the present bus value (including the current value of the invert bit) and the next data value.
- If the Hamming distance is larger than n/2, where n is the data bus width in bits, set the invert bit to one and make the next bus value equal to the inverted next data value.
- Otherwise, set the invert bit to zero and make the next bus value equal to the next data value.
- At the receiving end, the data must be decoded back to the original value based on the value on the invert bit.

The main disadvantages of DBI are:

- DBI is a fixed solution for all input data types.
- DBI is based on a very simple inverse transformation.
- DBI requires an extra bit line, i.e., hardware changes throughout the system.

The Bus Regrouping with Hamming Distance [8] method encodes data to reduce both the number of self and coupled transitions. A self-transition on the data bus is defined as a bit flip on the same bus line. A coupling transition on the data bus is defined as a bit flip between adjacent bus lines [8]. This method requires two extra bit lines. Its implementation is as follows:

- Calculate the number of coupled transitions and self-transitions on the data value and the bus value.
- If the number of coupled transitions is greater than half of the bus width, find the Hamming distance between the odd bits and the even bits in the data value and the bus value.
- If the odd hamming distance is greater than the even hamming distance, flip the data value in the odd bit positions and append a ‘1’ bit on the left and a ‘0’ bit on the right side of the encoded data.
- If the odd hamming distance is less than the even hamming distance, flip the data value in the even bit positions and append a ‘0’ bit on the left and a ‘1’ bit on the right side of the encoded data.
- If the odd hamming distance equals the even hamming distance, flip the entire data value and append a’1’ bit on the left as well as on the right side of the encoded data.

- If the coupled transitions are less than the bus width divided by two, transmits the data value as it is and append a ‘0’ bit on the left as well as on the right side of the encoded data.

The main disadvantage of this method is that it introduces two extra bit lines on the data bus.

The Limited Weight Code (LWC) and transition encoding [10] is a low-power encoding system to limit the number of 1’s on the data bus. This encoding assumes that the energy needed to transmit a 1 is greater than the energy to transmit a 0. DBI is a special case of LWC code where the number of 1’s is restricted to n/2 on an n bit input.

Bus Invert Transition Signaling (BITS) [11] is a low-power encoding technique for narrow buses. BITS is a combination of Bus Invert Coding [7] and transition signaling. In transition signaling, 1 is encoded as a transition (from low to high or from high to low) and 0 is encoded as a lack of transition. BITS requires an extra bit. Bus Invert Coding works best for randomly distributed data [11]. When the data patterns are not randomly distributed, as is often the case for telecommunication and image processing, the BITS algorithm performs better. BITS works as follows:

- If the number of 1’s in the data value is greater than n/2 (n is the data bus width), then each bit of data value is inverted (set invert bit =1) and then transition encoded. a transition (from high to low or from low to high) is encoded as 1 and then transition encoded.
- Otherwise, set the invert bit = 0 and each bit of the data value is transition encoded without any alterations.

- At the receiving end, the data must be decoded back to original value based on the value of the invert bit.

The main disadvantage of this method is that it is focused only on specific applications like speech and image processing, which often use narrow buses for transmission.

# 3. DESIGN AND IMPLEMENTATION

This chapter describes the design of ECL-BFM's automatically generated algorithms to minimize the number of bit flips on data.

The goal of encoding data is to transform input data into output or encoded form such that the number of bit flips between the previous and current data value is reduced. The reverse transformations are performed to decode the data.

To find effective bit-flip minimization algorithms for GPU data buses, I started with a detailed study of the Data Bus Inversion Algorithm and other previously proposed bit flip minimization algorithms/encodings and identified small individual algorithmic parts, which perform simple transformation on data. This yielded a number of algorithmic components for building effective algorithms. I use a tool called FLIPPY to implement each component using a common interface. Each component in FLIPPY can be given a block of data as input and transform it into an output block of data. This makes it possible to chain components together, thereby allowing us to generate a vast number of potential bit-flip minimization algorithms from a given set of components. Every component includes an inverse that performs the opposite transformation to get original data back. Therefore, for a given set of components, FLIPPY can generate a matching decoder.

I used FLIPPY to do an exhaustive search to determine the most effective bit-flip minimization algorithm from the given set of components. Based on a detailed analysis, I found the best one-, two-, and three-stage minimization algorithms for two types of input data, integer and floating-point, as well as the best overall algorithm that works for both input types.

Figure 2 represents a three-component encoder, where the last transformation is DBI. The decoder comprises of corresponding inverse components in reverse order to get back the original data.

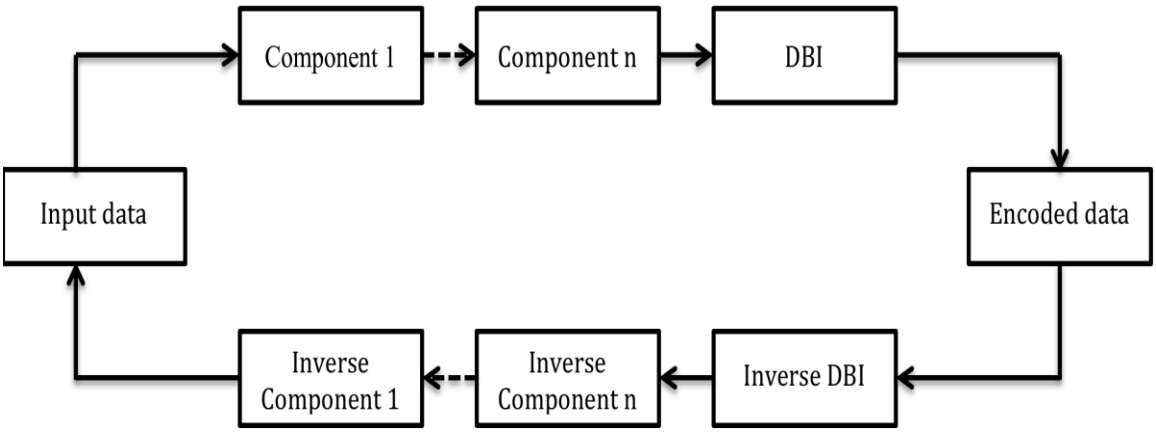


**Figure 2: n-component bit-flip minimization algorithm, where the last component is followed by DBI, and the corresponding inverse components that make up the decoding algorithm**

### 3.1 Algorithmic components

I identified the following algorithmic components for my thesis. Each component takes a block of data as input, transforms the data, and outputs it to the next component. All the algorithmic components work at byte granularity.

**NUL**

The NULL component does nothing to the data, it simply outputs the input data. Due to the presence of NUL component, the exhaustive search for three stage algorithm also includes all one- and two-stage algorithms. FLIPPY gives preference to shorter chains over longer chains.

**INV**

The INV component flips all the input bits.

**INVe**

The INVe component flips all even input bits.

**INVo**

The INVo component flips all odd input bits.

**SMS**

The SMS component converts each value from sign-magnitude (as used in the IEEE 754 floating-point format) into signed twos-complement representation. It does this by inverting all but the most significant bit if the most significant bit is set. This component may be important to reduce flips in signed data.

**BIT**

The BIT component shuffles the input data by creating a new sequence that contains the most significant bits of all bytes followed by the second most significant bits, etc.

**DIMn**

The DIMn takes an input n that specifies the dimensionality of the input sequence. For example, a dimension of three changes the linear sequence x1, y1, z1, x2, y2, z2, x3, y3, z3 into x1, x2, x3, y1, y2, y3, z1, z2, z3. In my thesis, n can take values of 2, 4, 8, 12, 16, 32, 64.

**ROTn**

The ROTn component takes a parameter n that specifies by how many units to rotate left the bits of each byte in the input sequence. There are seven versions of this component; n can take values from 1 to 7.

**GRAYl and GRAYr**

The GRAYd component converts the input bytes to Gray codes[12]. The GRAY component takes a parameter d, which is the direction at which the gray code is applied. The parameter d can take two values l (left) or r (right).

**LVs and LVx**

The LVs and LVx are predictors. Predictors guess the current value based on previous values in the input sequence, subtract the predicted from the current value, and emit the result of the subtraction, that is, the residual sequence. The subtraction to compute the residual sequence can be performed using byte level conventional subtraction (LVs) or at bit granularity using XOR (LVx).

**NEG**

The NEG component calculates the 2's complement of a number.

**BIp0**

The BIp0 checks the bit in position 0, if it is set, then inverts all other bits. It leaves the data as is if the bit is not set.

**BIn0**

The BIp0 checks the bit in position 0, if it is zero, then inverts all other bits. It leaves the data as is if the bit is set.

**PREp**

The PREp component calculates the number of bit flips in the previous byte. If there are more than four flips, it inverts all the bits in the current byte. Otherwise, it leaves the current byte as it is.

**PREn**

The PREn component calculates the number of bit flips in the previous byte. If there are fewer than four flips, it inverts all the bits in the current byte. Otherwise, it leaves the current byte as is.

Every component in FLIPPY has an inverse component that performs the inverse transformation to get back the original data. FLIPPY has 29 components in total excluding DBI. It evaluates 24,389 combinations by chaining three components together to find the best minimization algorithm for a given input data.

Since FLIPPY has 29 algorithmic components, performing an exhaustive search to identify the best algorithm is too expensive for chains with more than three components, i.e., algorithms with more than three stages.

# 4. EVALUATION METHODS

## 4.1 Generation of traces used for evaluation

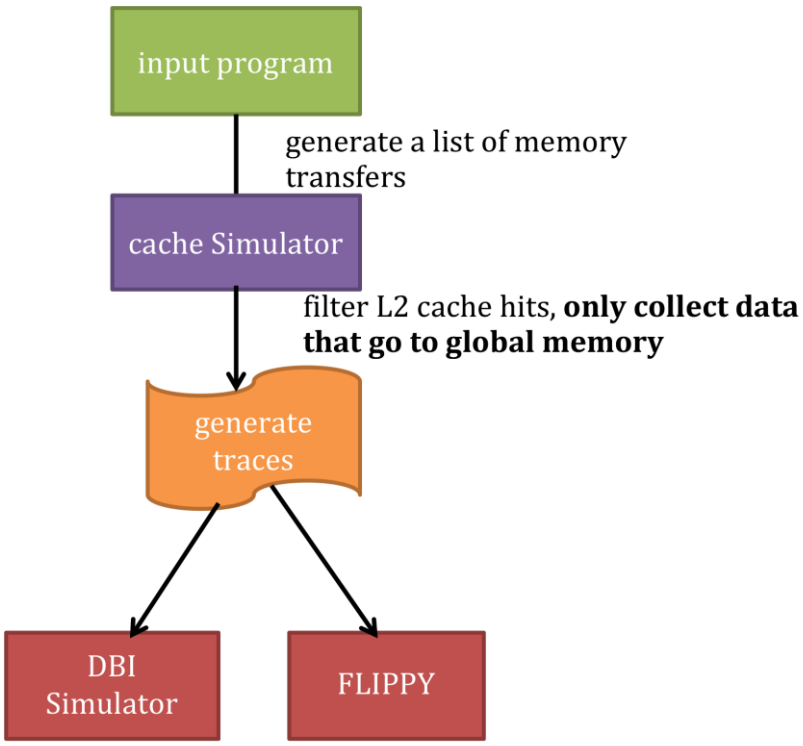


**Figure 3: Generation of input traces used for evaluation**

Figure 3 shows the generation of traces used for evaluation. The first step is to collect all data transfers between memory and GPU processor. Next the collected data is passed through a cache simulator. The cache simulator removes all data transfers that hit in the cache and only emits cache misses and write back transfers to the main memory. The output from the cache simulator is stored in a trace file, which contains all data transfers on the memory bus for a given program. The generated trace file is used by the FLIPPY tool and by a DBI simulator for evaluation.

### 4.2 Traces

I collected 30 trace files for the evaluation. Each trace file contains all memory data transferred between the GPU and off-chip memory during a program execution. In other words, a trace file contains all data transfers on the data bus to and from global memory. GPUs have different memory spaces: global, cache, shared, and texture. This thesis focuses on data transfers between the GPU processor and global memory. I used the following steps to collect the traces.

- Collect a list of all data transfers on data bus during the program execution.
- Pass the collected list to a cache simulator. The output from the cache simulator only contains cache miss and write back transfers that go to the GPU's global memory.
- The cache simulator output is put in a trace file, which is used for the evaluation.

### 4.3 Memory trace function

In GPUs, 32 threads are executed in parallel, which is called a "warp". All memory access are split into 128 byte transfers, how many ever transfers are needed.

To collect these 128-byte memory transfers, I wrote a memory trace function in CUDA, called by all threads in a warp. The function takes memory address and type of transfer as input. The active threads in a warp are calculated using __ballot() and __ffs() functions. The lowest active thread in the warp is used to collect all 128 bytes of data for all memory accesses.

### 4.4 Cache simulation

A cache is a fast and small memory between the processor and global memory that stores some of the most recently accessed data. Most GPUs contains two levels of caches, L1 cache and L2 cache. In this thesis I used two 4-way set associative caches of size 16KB (L1 cache) and 2MB(L2 cache) to filter out the cache hits and record only data transfers of accesses that go to global memory. Output from the cache simulator is called as a trace file.

The trace files from L1 cache and L2 cache were similar and L2 trace files also include L1 miss and write-backs that go to global memory. Therefore, for the algorithm evaluation I used the trace file generated from L2 cache.

### 4.5 Input programs

I used the following algorithms to collect traces for the evaluation.

Integer programs:

- Fractal - Mandelbrot Set

  Mandelbrot set is a set of complex numbers c for which the sequence $z_{n+1} = z_n^2 + c$; $z_0 = 0$ is bounded. This set creates a complex pattern that is called “fractal”. The number of iterations before reaching a given maximum determines the brightness of each pixel in the fractal.

- Maximal Independent Set (MIS)

  The MIS algorithm calculates the maximal independent sets for a given input graph based on Luby’s algorithm. We use three input graphs to collect traces from MIS.

- Delaunay Mesh Refinement (DMR)

A 2D Delaunay mesh is a triangularization of a set of points with the following property: the circumcircle of any triangle in the mesh must contain no other point from the mesh. A refined Delaunay mesh is a Delaunay mesh with the additional constraint that no triangle have an angle of less than 30 degrees. The algorithm takes an input Delaunay mesh, containing triangles that contain angles that do not meet the above constraints and produces a refined mesh by iteratively re-triangulating the affected portions of the mesh.

- Minimum Spanning Tree (MST)

The MST algorithm calculates a subset of edges that connects all the vertices together. This algorithm computes a minimum spanning tree in a weighted graph using Boruvka's algorithm.

- Massively Parallel Compression (MPC)

MPC is a parallel data compression algorithm for scientific data. It is a GPU-based compressor for single- and double-precision floating-point values.

Floating-point programs:

- N-Body Simulation

N-body simulation is a simulation of stars under the influence of physical forces like gravity.

- Barnes Hut Algorithm

This algorithm simulates the gravitational forces acting on a galactic cluster using the Barnes-Hut n-body algorithm. The program calculates the motion of each star through space for a number of time steps.

- Travelling Salesman Problem (TSP)

This graph algorithm uses a heuristic to find a short route that visits each set of points exactly once.

- Binomial Options

  This algorithm evaluates the fair call price for a given set of European options under a binomial model.

- Fast Walsh Transform

  This algorithm simulates the naturally (hadamard)-ordered Fast Walsh Transform for batching vectors of arbitrary eligible lengths that are power of two in size.

### 4.6 Measuring bit flips

To measure the number of bit flips on the GPU data bus, we created a data bus simulator. The width of the bus is 64 bits. Data is transferred in 128 byte chunks (32 words), this is same as the cache block size in GPUs. The bus is initialized to zero at the start of the simulation. The bit flips are counted between the current value (to be transmitted over the bus) and the previous value (on the bus) using the popcount function.

### 4.7 DBI simulation

For my thesis, I created a DBI simulation to compare my algorithms to DBI. The implementation is written in C. I implemented DBI for a 64 bit data bus. The 64 bit data bus is divided into eight 8-bit data buses with an extra bit line for each byte. Thus, the 64 bit input data from the trace is split into eight 8-bit parallel data sets. If there are more than four bit flips between the previous data and the current data, then the extra bit is set and the current data is inverted before sending it over the bus. Otherwise, the data is sent as it is.

# 5. RESULTS

This chapter presents the main results. The results are divided based on number of algorithmic components as well as integer and floating point input data. It also includes the customized bit flip reduction algorithm for a specific input data.

## 5.1 Bit-flip minimization algorithms with extra bit line

### 5.1.1. One-component algorithm

Based on an exhaustive search using the 29 algorithmic components in FLIPPY, the most efficient one-component bit-flip minimization algorithm with DBI for all input traces is DIM64 DBI. The DIM64 component rearranges the data in such a way that the interval between two consecutive bytes is 64, i.e., byte 0 and byte 64 are grouped together, then byte 1 and byte 65, etc., in each 128-byte chunk. DBI is then applied to the transformed data. The algorithm uses the DIM64 component to separate different byte positions. This component is useful for both integers and floating point values, as their topmost bits are often the same. By grouping them together, DBI becomes more effective.

### 5.1.2. Two-component algorithm

The most efficient two-component algorithm for all input traces is GRAY DIM64 DBI. The GRAY component converts the input into Gray code [12], in which the number of bit changes between two consecutive integers is always one. The output of the GRAY component is the input to DIM64, which is explained above. Chaining the two components together further reduces the bit flips. The addition of GRAY component is useful for all inputs, particularly floating-point data as their topmost bits in Gray code is

grouped together by DIM64 to reduce 9% more bit flips than DBI and 1.5% more bit flips than one-component algorithm.

### 5.1.3. Three-component algorithm

The best three-component algorithms for all input traces is SMS GRAY DIM64 DBI. The SMS component inverts the seven least significant bits in a byte if the most significant bit is set. If it is not set, the remaining bits are not inverted. The output of the SMS component is the input to GRAY. The GRAY component transforms the data into Gray code [12], which is the input for the DIM64 component. The SMS component specifically helps to reduce bit flips in sign-magnitude data, i.e., floating-point representation. The addition of the SMS component results in a 0.5% higher bit-flip reduction than the best two-component algorithm.

### 5.1.4. The reduction of bit flips using ECL-BFM

The below graph in Figure 4, shows the average reduction in bit flips compared to the existing solution of Data Bus Inversion. The first bar in blue represents the average number of bit flips in the input files, i.e., the inputs files naturally contain 27% bit flips. The second bar shows the average reduction in bit flips after applying DBI. DBI reduces the bit flips to about 21.79%. My solution reduces the bit flips up to 19.65%. In other words, my algorithm reduces 9% more bit flips than DBI.

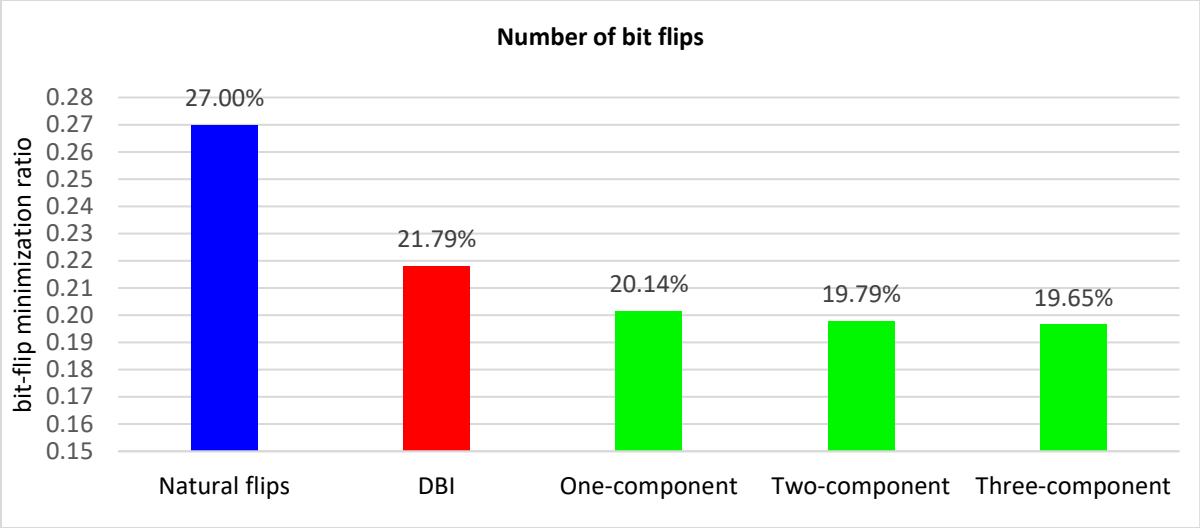


**Figure 4: Total number of bit flips (%)**

We can see that the addition of the SMS component to the two-stage algorithm yields only a small reduction in bit flips over the two-component algorithm. Therefore, I believe the two-component algorithm GRAY DIM64 DBI to be the most efficient and cost-effective solution to reduce bit flips across integer and floating-point inputs.

### 5.1.5. Customized bit-flip minimization algorithm for a given input

Based on an exhaustive search using FLIPPY, I found that the two-component algorithm GRAY DIM64 | DBI works very well across all inputs. However, for some of the inputs, I was able to identify an algorithm that works much better than GRAY DIM64 | DBI. This algorithm is the Customized bit-flip-reduction algorithm FLIPPY can find for a given input. However, this solution is potentially different for every input trace. Tables 1, 2, and 3 show the maximum bit-flip reduction algorithm (for one, two, and three components) for all tested inputs.

**Table 1. Bit flip minimization ratios of DBI and ECL-BFM one-component algorithm**

| Input trace | DBI | DIM64 DBI | Customized algorithm for the input | Customized algorithm |
|---|---|---|---|---|
| | bit flip minimization ratio | bit flip minimization ratio | | bit flip minimization ratio |
| Fractal_30_512 | 1.189 | 1.509 | DIM16 DBI | 1.971 |
| Fractal_60_256 | 1.196 | 1.458 | DIM16 DBI | 2.077 |
| Fractal_60_512 | 1.191 | 1.461 | DIM16 DBI | 2.070 |
| MIS_internet | 1.233 | 1.333 | DIM64 DBI | 1.333 |
| MIS_amazon | 1.189 | 1.369 | DIM64 DBI | 1.369 |
| MIS_USAroadmap | 1.166 | 1.220 | DIM64 DBI | 1.220 |
| MPC_7770102_10 | 1.294 | 1.403 | DIM64 DBI | 1.403 |
| MPC_13418496_10 | 1.225 | 1.266 | DIM64 DBI | 1.266 |
| MPC_33298679_1 | 1.243 | 1.271 | DIM64 DBI | 1.271 |
| MST_rmat12 | 1.166 | 1.160 | NUL DBI | 1.166 |
| MST_2d-2e20 | 1.148 | 1.148 | NUL DBI | 1.148 |
| MST_USA-road | 1.153 | 1.206 | DIM64 DBI | 1.206 |
| DMR_input1 | 1.193 | 1.127 | NUL DBI | 1.193 |
| DMR_input2 | 1.223 | 1.310 | DIM64 DBI | 1.310 |
| DMR_input3 | 1.193 | 1.125 | NUL DBI | 1.193 |
| Binomialoptions_input1 | 1.299 | 1.415 | DIM64 DBI | 1.415 |
| Binomialoptions_input2 | 1.304 | 1.413 | DIM64 DBI | 1.413 |
| Binomialoptions_input3 | 1.302 | 1.409 | DIM64 DBI | 1.409 |
| Nbody_10000_10 | 1.205 | 1.205 | DIM64 DBI | 1.205 |
| Nbody_13500_10 | 1.205 | 1.205 | DIM64 DBI | 1.205 |
| Nbody_26650_10 | 1.206 | 1.205 | INV DBI | 1.206 |
| TSP_225_1000 | 1.177 | 1.282 | DIM64 DBI | 1.282 |
| TSP_575_100 | 1.174 | 1.218 | DIM64 DBI | 1.218 |
| TSP_100_10000 | 1.189 | 1.241 | DIM64 DBI | 1.241 |
| FWT_input1 | 1.191 | 1.192 | DIM64 DBI | 1.192 |
| FWT_input2 | 1.305 | 1.310 | DIM4 DBI | 1.603 |
| FWT_input3 | 1.191 | 1.192 | DIM64 DBI | 1.192 |
| BH_10000_100 | 1.397 | 1.444 | DIM64 DBI | 1.444 |
| BH_30000_50 | 1.396 | 1.443 | DIM64 DBI | 1.443 |
| BH_15000_25 | 1.422 | 1.473 | DIM64 DBI | 1.473 |

**Table 2. Bit flip minimization ratios of DBI and ECL-BFM two-component algorithm**

| Input trace | DBI | GRAY DIM64 DBI | Customized algorithm for the input | Customized algorithm |
|---|---|---|---|---|
| | bit flip minimization ratio | bit-flip minimization ratio | | bit-flip minimization ratio |
| Fractal_30_512 | 1.196 | 1.645 | GRAYr DIM16 DBI | 2.106 |
| Fractal_60_256 | 1.191 | 1.579 | GRAYr DIM16 DBI | 2.242 |
| Fractal_60_512 | 1.189 | 1.574 | GRAYr DIM16 DBI | 2.237 |
| MIS_internet | 1.233 | 1.391 | GRAYr DIM64 DBI | 1.391 |
| MIS_amazon | 1.166 | 1.250 | GRAYr DIM64 DBI | 1.250 |
| MIS_USAroadmap | 1.189 | 1.407 | GRAYr DIM64 DBI | 1.407 |
| MPC_7770102_10 | 1.294 | 1.280 | GRAYr DIM64 DBI | 1.280 |
| MPC_13418496_10 | 1.243 | 1.298 | GRAYr DIM64 DBI | 1.298 |
| MPC_33298679_1 | 1.225 | 1.430 | NEG DIM64 DBI | 1.438 |
| MST_rmat12 | 1.148 | 1.178 | GRAYr DIM64 DBI | 1.178 |
| MST_2d-2e20 | 1.166 | 1.178 | GRAYr DIM64 DBI | 1.178 |
| MST_USA-road | 1.153 | 1.281 | GRAYr DIM64 DBI | 1.281 |
| DMR_input1 | 1.193 | 1.314 | GRAYr DIM64 DBI | 1.314 |
| DMR_input2 | 1.223 | 1.113 | NUL NUL DBI | 1.223 |
| DMR_input3 | 1.193 | 1.111 | NUL NUL DBI | 1.193 |
| Binomialoptions_input1 | 1.299 | 1.443 | GRAYr DIM64 DBI | 1.443 |
| Binomialoptions_input2 | 1.302 | 1.447 | GRAYr DIM64 DBI | 1.447 |
| Binomialoptions_input3 | 1.304 | 1.440 | GRAYr DIM64 DBI | 1.440 |
| Nbody_10000_10 | 1.205 | 1.159 | INV NEG DBI | 1.210 |
| Nbody_13500_10 | 1.206 | 1.160 | INV NEG DBI | 1.211 |
| Nbody_26650_10 | 1.205 | 1.160 | INV NEG DBI | 1.211 |
| TSP_225_1000 | 1.189 | 1.211 | NUL DIM64 DBI | 1.241 |
| TSP_575_100 | 1.177 | 1.184 | NUL DIM64 DBI | 1.218 |
| TSP_100_10000 | 1.174 | 1.297 | GRAYl DIM64 DBI | 1.317 |
| FWT_input1 | 1.191 | 1.182 | ROTl6 NEG DBI | 1.219 |
| FWT_input2 | 1.191 | 1.293 | GRAYr DIM4 DBI | 1.650 |
| FWT_input3 | 1.305 | 1.182 | ROTl6 NEG DBI | 1.219 |
| BH_10000_100 | 1.397 | 1.411 | NUL DIM64 DBI | 1.444 |
| BH_30000_50 | 1.396 | 1.411 | NUL DIM64 DBI | 1.443 |
| BH_15000_25 | 1.422 | 1.440 | NUL DIM64 DBI | 1.473 |

**Table 3. Bit flip minimization ratios of DBI and ECL-BFM three-component algorithm**

| Input trace | DBI | SMS GRAY DIM64 DBI | Customized algorithm for the input | Customized algorithm |
|---|---|---|---|---|
| | bit-flip minimization ratio | bit-flip minimization ratio | | bit- flip minimization ratio |
| Fractal_30_512 | 1.196 | 2.106 | GRAYr BIn0 DIM16 DBI | 2.242 |
| Fractal_60_256 | 1.191 | 2.242 | NEG GRAYr DIM16 DBI | 2.245 |
| Fractal_60_512 | 1.189 | 2.237 | NEG GRAYr DIM16 DBI | 2.107 |
| MIS_internet | 1.233 | 1.391 | GRAYr BIn0 DIM64 DBI | 1.391 |
| MIS_amazon | 1.166 | 1.250 | DIM4 LVs DIM32 DBI | 1.279 |
| MIS_USAroadmap | 1.189 | 1.407 | GRAYr BIn0 DIM64 DBI | 1.407 |
| MPC_7770102_10 | 1.294 | 1.280 | SMS NEG DIM64 DBI | 1.440 |
| MPC_13418496_10 | 1.243 | 1.298 | DIM4 LVs DIM32 DBI | 1.315 |
| MPC_33298679_1 | 1.225 | 1.438 | GRAYr BIn0 DIM64 DBI | 1.280 |
| MST_rmat12 | 1.148 | 1.178 | DIM16 BIT DIM4 DBI | 1.263 |
| MST_2d-2e20 | 1.166 | 1.178 | BIn0 ROTl7 GRAYr DBI | 1.185 |
| MST_USA-road | 1.153 | 1.281 | GRAYr BIn0 DIM64 DBI | 1.281 |
| DMR_input1 | 1.193 | 1.314 | NEG ROTl2 NEG DBI | 1.211 |
| DMR_input2 | 1.223 | 1.193 | GRAYr GRAYr DIM64 DBI | 1.326 |
| DMR_input3 | 1.193 | 1.193 | NEG ROTl2 NEG DBI | 1.210 |
| Binomialoptions_input1 | 1.299 | 1.443 | BIn0 NEG DIM64 DBI | 1.457 |
| Binomialoptions_input2 | 1.302 | 1.447 | DIM16 BIT DIM16 DBI | 1.455 |
| Binomialoptions_input3 | 1.304 | 1.440 | DIM16 BIT DIM16 DBI | 1.462 |
| Nbody_10000_10 | 1.205 | 1.210 | INV ROTl7 NEG DBI | 1.210 |
| Nbody_13500_10 | 1.206 | 1.211 | INV NEG BIn0 DBI | 1.211 |
| Nbody_26650_10 | 1.205 | 1.211 | INV NEG BIn0 DBI | 1.211 |
| TSP_225_1000 | 1.189 | 1.241 | INV NEG DIM64 DBI | 1.253 |
| TSP_575_100 | 1.177 | 1.218 | BIn0 NEG DIM64 DBI | 1.326 |
| TSP_100_10000 | 1.174 | 1.317 | BIn0 ROTl1 DIM64 DBI | 1.218 |
| FWT_input1 | 1.191 | 1.219 | ROTl6 NEG DIM64 DBI | 1.220 |
| FWT_input2 | 1.191 | 1.650 | ROTl6 NEG DIM64 DBI | 1.220 |
| FWT_input3 | 1.305 | 1.219 | GRAYr DIM4 BIT DBI | 1.665 |
| BH_10000_100 | 1.397 | 1.444 | INV NEG DIM64 DBI | 1.447 |
| BH_30000_50 | 1.396 | 1.443 | INV NEG DIM64 DBI | 1.446 |
| BH_15000_25 | 1.422 | 1.473 | INV NEG DIM64 DBI | 1.481 |

The bit-flip minimization ratio in tables 1, 2, 3 represent the number of existing bit flips on the input divided by the number of bit flips after applying the algorithm. It can be observed that the customized algorithm for a given input can reduce an average of 11% higher bit flips than DBI. For some integer inputs like fractal, the customized algorithm is DIM16 DBI, which reduces 47% more bit-flips than DBI using just a one-component algorithm.

The customized two-component algorithm for a given input can reduce up to 14% bit flips on an average than DBI. Therefore, if the nature of the input is known in advance, then my solution using the customized algorithm for that specific input can reduce significantly more bit flips than DBI.

The below graphs shows the summary of results for one-, two-, and three-component algorithms. The graph compares the bit flip minimization ratios of DBI and ECL-BFM algorithms for all input programs. The first five programs are integer type and the last five programs are floating-point type. Each input program was run with three inputs and the average performance of algorithms for three inputs is plotted. The last bar graph shows the geometric mean of DBI and ECL-BFM algorithms.

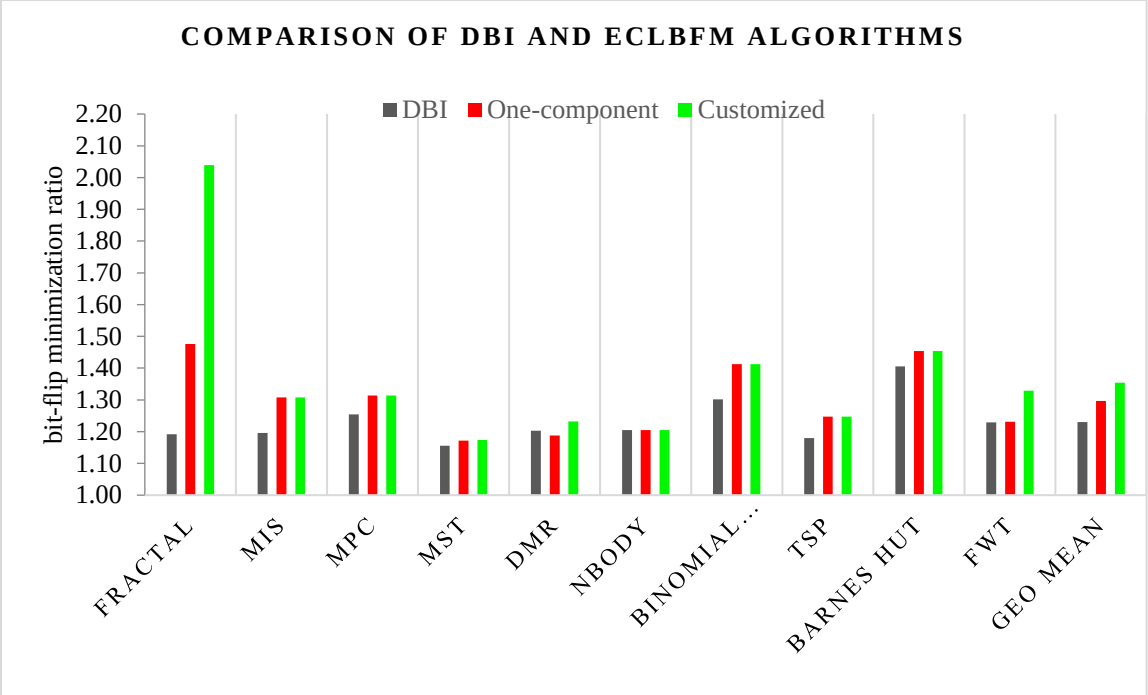


**Figure 5: Comparison between DBI and ECL-BFM one-component algorithm**

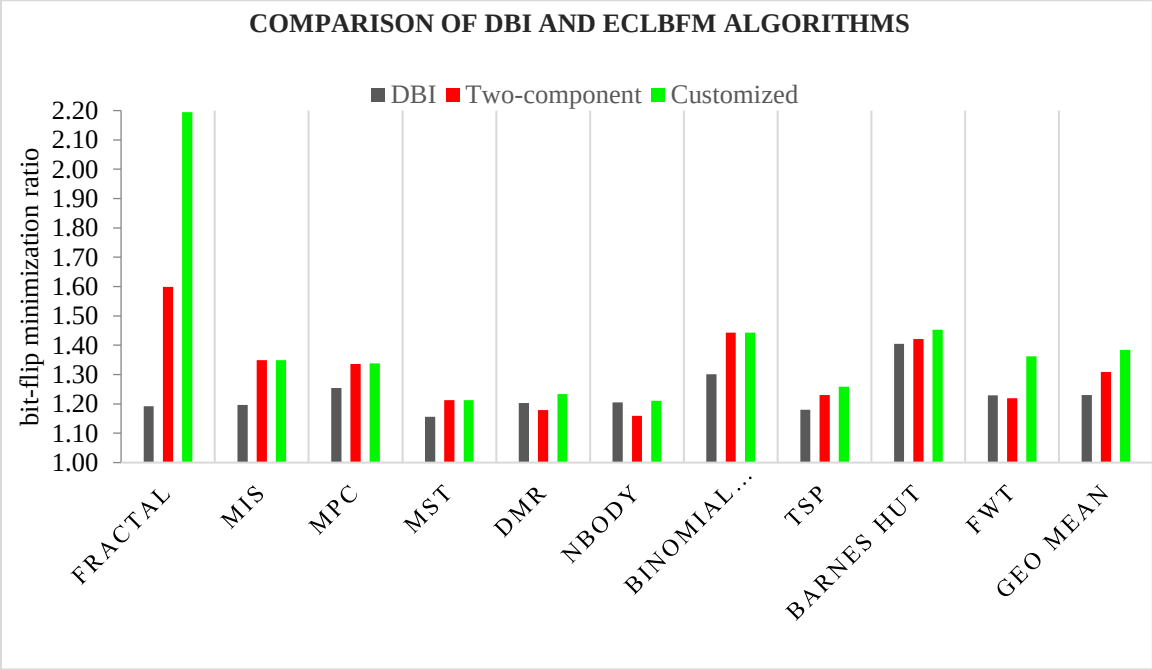


**Figure 6: Comparison between DBI and ECL-BFM two-component algorithm**

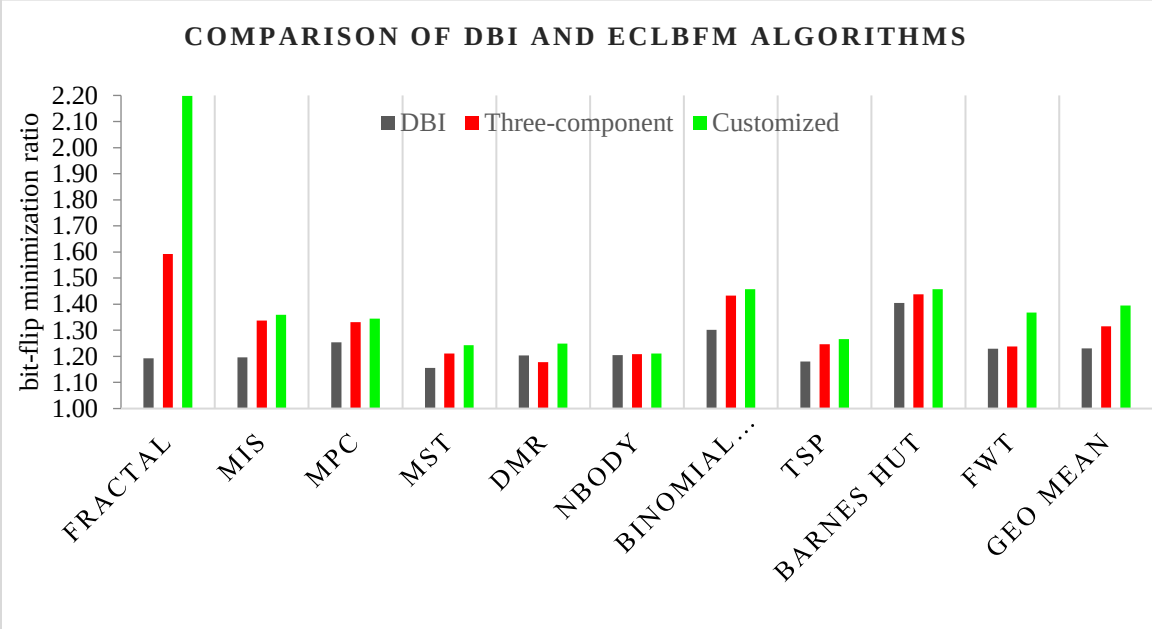


**Figure 7: Comparison between DBI and ECL-BFM three-component algorithm**

## 5.2 Bit-flip minimization algorithm without extra bit line

This thesis also proposes a low-cost solution to reduce bit flips without the need for an extra bit line.

### 5.2.1. One-component algorithm

Based on an exhaustive search using the 29 algorithmic components in FLIPPY without an extra bit line, the most efficient one-component bit-flip minimization algorithm for all input traces is DIM64. The DIM64 component performs the transformation explained above before placing the data onto the bus.

### 5.2.2. Two-component algorithm

The most efficient two-component algorithm for all input traces is GRAY DIM64. These two components perform the transformation explained above.

### 5.2.3. Three-component algorithm

The best three-component algorithm for all input traces is SMS GRAY DIM64. Again, this is the same algorithm as above when it is coupled with DBI.

### 5.2.4. The reduction of bit flips using ECL-BFM algorithms without extra bit line

The below graph in Figure 4 shows the total number of bit flips compared to the existing solution, Data Bus Inversion. The first bar in blue represents the average number of bit flips in input files, i.e., 27%. The second bar shows the number of bit flips after applying DBI. The DBI reduces the bit-flips in a given file to about 21.79%. My solution reduces the overall bit flips to about ~24% but without the need for an extra bit line. In other words, my algorithm eliminates 11% of the bit flips without using the extra bit line.

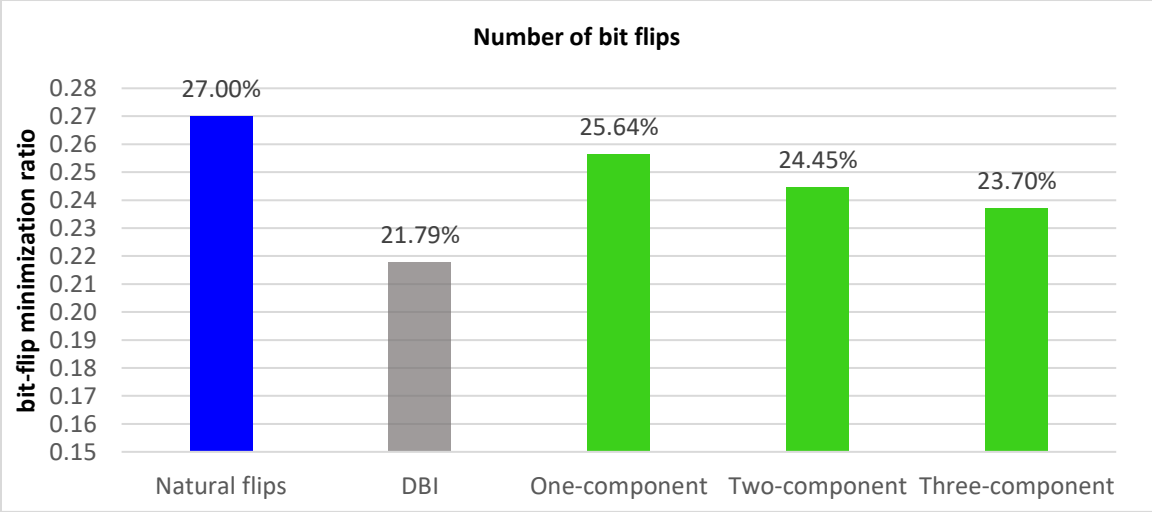


**Figure 8: Total number of bit flips (%)**

We can see that the addition of the SMS component to the two-stage algorithm reduces 3% more bit flips than the two-component algorithm. Therefore, the three-component algorithm SMS GRAY DIM64 is probably the most cost effective solution to reduce bit flips for all integer and floating-point input types when no extra bit line is available.

### 5.2.5. Customized bit-flip minimization algorithm for a given input

The tables 4, 5, 6 show the customized bit flip-reduction algorithm with one-, two-, and three-components for all input traces without the need for an extra bit.

**Table 4. Bit flip minimization ratios of DBI and ECL-BFM one-component algorithm**

| Input file | DBI | DIM64 | Customized algorithm for the input | Customized algorithm |
|---|---|---|---|---|
| | bit flip minimization ratio | bit flip minimization ratio | | bit- flip minimization ratio |
| Fractal_30_512 | 1.189 | 1.286 | DIM16 | 1.697 |
| Fractal_60_256 | 1.196 | 1.233 | DIM16 | 1.786 |
| Fractal_60_512 | 1.191 | 1.242 | DIM16 | 1.787 |
| MIS_internet | 1.233 | 1.084 | GRAYr | 1.090 |
| MIS_amazon | 1.189 | 1.159 | DIM64 | 1.047 |
| MIS_USAroadmap | 1.166 | 1.047 | DIM64 | 1.159 |
| MPC_7770102_10 | 1.294 | 1.080 | NEG | 1.087 |
| MPC_13418496_10 | 1.225 | 1.014 | GRAYr | 1.046 |
| MPC_33298679_1 | 1.243 | 1.024 | GRAYr | 1.032 |
| MST_rmat12 | 1.166 | 1.000 | NUL | 1.000 |
| MST_2d-2e20 | 1.148 | 1.048 | NUL | 1.000 |
| MST_USA-road | 1.153 | 0.994 | DIM64 | 1.048 |
| DMR_input1 | 1.193 | 0.939 | SMS | 1.003 |
| DMR_input2 | 1.223 | 1.073 | DIM64 | 1.073 |
| DMR_input3 | 1.193 | 0.937 | SMS | 1.001 |
| Binomialoptions_input1 | 1.299 | 1.083 | DIM16 | 1.134 |
| Binomialoptions_input2 | 1.304 | 1.080 | DIM16 | 1.139 |
| Binomialoptions_input3 | 1.302 | 1.083 | DIM16 | 1.134 |
| Nbody_10000_10 | 1.205 | 1.000 | NUL | 1.000 |
| Nbody_13500_10 | 1.205 | 0.999 | NUL | 1.000 |
| Nbody_26650_10 | 1.206 | 1.000 | NUL | 1.000 |
| TSP_225_1000 | 1.177 | 1.090 | DIM64 | 1.090 |
| TSP_575_100 | 1.174 | 1.039 | DIM64 | 1.039 |
| TSP_100_10000 | 1.189 | 1.050 | DIM64 | 1.050 |
| FWT_input1 | 1.191 | 1.001 | NEG | 1.158 |
| FWT_input2 | 1.305 | 1.001 | NEG | 1.157 |
| FWT_input3 | 1.191 | 1.032 | GRAYr | 1.175 |
| BH_10000_100 | 1.397 | 1.093 | DIM64 | 1.001 |
| BH_30000_50 | 1.396 | 1.092 | DIM64 | 1.001 |
| BH_15000_25 | 1.422 | 1.101 | DIM4 | 1.276 |

**Table 5. Bit flip minimization ratios of DBI and ECL-BFM two-component algorithm**

| Input file | DBI | GRAY DIM64 | Customized algorithm for the input | Customized algorithm |
|---|---|---|---|---|
| | bit-flip minimization ratio | bit-flip minimization ratio | | bit- flip minimization ratio |
| Fractal_30_512 | 1.189 | 1.448 | GRAYr DIM16 | 1.849 |
| Fractal_60_256 | 1.196 | 1.385 | GRAYr DIM16 | 1.993 |
| Fractal_60_512 | 1.191 | 1.389 | GRAYr DIM16 | 1.998 |
| MIS_internet | 1.233 | 1.216 | GRAYr DIM64 | 1.216 |
| MIS_amazon | 1.166 | 1.218 | GRAYr DIM64 | 1.218 |
| MIS_USAroadmap | 1.189 | 1.116 | GRAYr DIM64 | 1.116 |
| MPC_7770102_10 | 1.225 | 1.178 | NEG DIM64 | 1.186 |
| MPC_13418496_10 | 1.243 | 1.068 | GRAYr DIM64 | 1.068 |
| MPC_33298679_1 | 1.294 | 1.069 | GRAYr DIM64 | 1.069 |
| MST_rmat12 | 1.148 | 1.008 | GRAYr DIM64 | 1.008 |
| MST_2d-2e20 | 1.166 | 1.016 | GRAYr DIM64 | 1.016 |
| MST_USA-road | 1.153 | 1.104 | GRAYr DIM64 | 1.104 |
| DMR_input1 | 1.223 | 0.928 | SMS INV | 1.003 |
| DMR_input2 | 1.193 | 1.086 | GRAYr DIM64 | 1.086 |
| DMR_input3 | 1.193 | 0.925 | SMS INV | 1.001 |
| Binomialoptions_input1 | 1.304 | 1.226 | GRAYr DIM64 | 1.226 |
| Binomialoptions_input2 | 1.299 | 1.220 | GRAYr DIM64 | 1.220 |
| Binomialoptions_input3 | 1.302 | 1.219 | GRAYr DIM64 | 1.219 |
| Nbody_10000_10 | 1.205 | 0.976 | SMS GRAYr | 1.011 |
| Nbody_13500_10 | 1.205 | 0.976 | SMS GRAYr | 1.011 |
| Nbody_26650_10 | 1.206 | 0.976 | SMS GRAYr | 1.011 |
| TSP_225_1000 | 1.189 | 1.041 | DIM64 BIT | 1.090 |
| TSP_575_100 | 1.174 | 1.040 | GRAYr DIM64 | 1.040 |
| TSP_100_10000 | 1.177 | 1.064 | NEG DIM64 | 1.075 |
| FWT_input1 | 1.191 | 0.990 | SMS GRAYr | 1.006 |
| FWT_input2 | 1.305 | 1.091 | GRAYr DIM4 | 1.336 |
| FWT_input3 | 1.191 | 0.990 | SMS GRAYr | 1.006 |
| BH_10000_100 | 1.397 | 1.201 | NEG DIM64 | 1.195 |
| BH_30000_50 | 1.396 | 1.200 | GRAYr DIM64 | 1.219 |
| BH_15000_25 | 1.422 | 1.220 | NEG DIM64 | 1.196 |

**Table 6. Bit flip minimization ratios of DBI and ECL-BFM three-component algorithm**

| Input file | DBI | SMS GRAY DIM64 | Customized algorithm for the input | Customized algorithm |
|---|---|---|---|---|
| | bit flip minimization ratio | bit flip minimization ratio | | bit flip minimization ratio |
| Fractal_30_512 | 1.196 | 1.448 | GRAYr DIM16 BIT | 1.849 |
| Fractal_60_256 | 1.191 | 1.385 | GRAYr DIM16 BIT | 1.993 |
| Fractal_60_512 | 1.189 | 1.389 | LVs BIT DIM16 | 2.013 |
| MIS_internet | 1.233 | 1.216 | GRAYr DIM64 BIT | 1.216 |
| MIS_amazon | 1.166 | 1.218 | GRAYr DIM64 BIT | 1.218 |
| MIS_USAroadmap | 1.189 | 1.116 | GRAYr SMS DIM64 | 1.125 |
| MPC_7770102_10 | 1.294 | 1.178 | ROTl1 NEG DIM64 | 1.186 |
| MPC_13418496_10 | 1.243 | 1.068 | GRAYr DIM64 BIT | 1.068 |
| MPC_33298679_1 | 1.225 | 1.069 | GRAYr DIM64 BIT | 1.069 |
| MST_rmat12 | 1.148 | 1.008 | BIp0 ROTl7 GRAYr | 1.015 |
| MST_2d-2e20 | 1.166 | 1.016 | DIM64 BIT GRAYr | 1.036 |
| MST_USA-road | 1.153 | 1.104 | GRAYr DIM64 BIT | 1.107 |
| DMR_input1 | 1.193 | 0.928 | GRAYr INVe GRAYl | 1.012 |
| DMR_input2 | 1.223 | 1.086 | GRAYr DIM64 BIT | 1.090 |
| DMR_input3 | 1.193 | 0.925 | GRAYr INVe GRAYl | 1.011 |
| Binomialoptions_input1 | 1.299 | 1.209 | GRAYr DIM64 BIT | 1.226 |
| Binomialoptions_input2 | 1.302 | 1.205 | GRAYr DIM64 BIT | 1.220 |
| Binomialoptions_input3 | 1.304 | 1.204 | GRAYr DIM64 BIT | 1.219 |
| Nbody_10000_10 | 1.205 | 1.011 | SMS GRAYr DIM64 | 1.011 |
| Nbody_13500_10 | 1.206 | 1.011 | SMS GRAYr DIM64 | 1.011 |
| Nbody_26650_10 | 1.205 | 1.011 | SMS GRAYr DIM64 | 1.011 |
| TSP_225_1000 | 1.189 | 1.048 | GRAYr GRAYr DIM64 | 1.134 |
| TSP_575_100 | 1.177 | 1.055 | SMS GRAYr DIM64 | 1.055 |
| TSP_100_10000 | 1.174 | 1.068 | NEG DIM64 BIT | 1.075 |
| FWT_input1 | 1.191 | 1.006 | SMS GRAYr DIM64 | 1.009 |
| FWT_input2 | 1.191 | 1.116 | GRAYr BIT DIM4 | 1.402 |
| FWT_input3 | 1.305 | 1.006 | SMS GRAYr DIM64 | 1.009 |
| BH_10000_100 | 1.397 | 1.201 | SMS GRAYr DIM64 | 1.204 |
| BH_30000_50 | 1.396 | 1.200 | SMS GRAYr DIM64 | 1.204 |
| BH_15000_25 | 1.422 | 1.220 | ROTl2 SMS DIM64 | 1.228 |

It can be observed that the customized algorithm for unsigned integer inputs like fractal can reduce up to 47% more bit flips than DBI without the need for an extra bit and even though DBI uses an extra bit. For some of the integer inputs, the best performing algorithm using my solution reduces almost as many bit flips as DBI but without the need of any additional cost in terms of extra bits (one per byte on the bus). However, for floating-point inputs, DBI performs 10% better than my best algorithm.

The below graphs summarize the results for the one-, two-, and three-component algorithms. The graph compares the bit-flip minimization ratios of DBI and the ECL-BFM algorithms for all input programs. The first five programs are of integer type and the last five programs are of floating-point type. Each input program was run with three inputs and the average performance of algorithms for three inputs is plotted. The last bar graph shows the geometric mean of DBI and ECL-BFM algorithms.

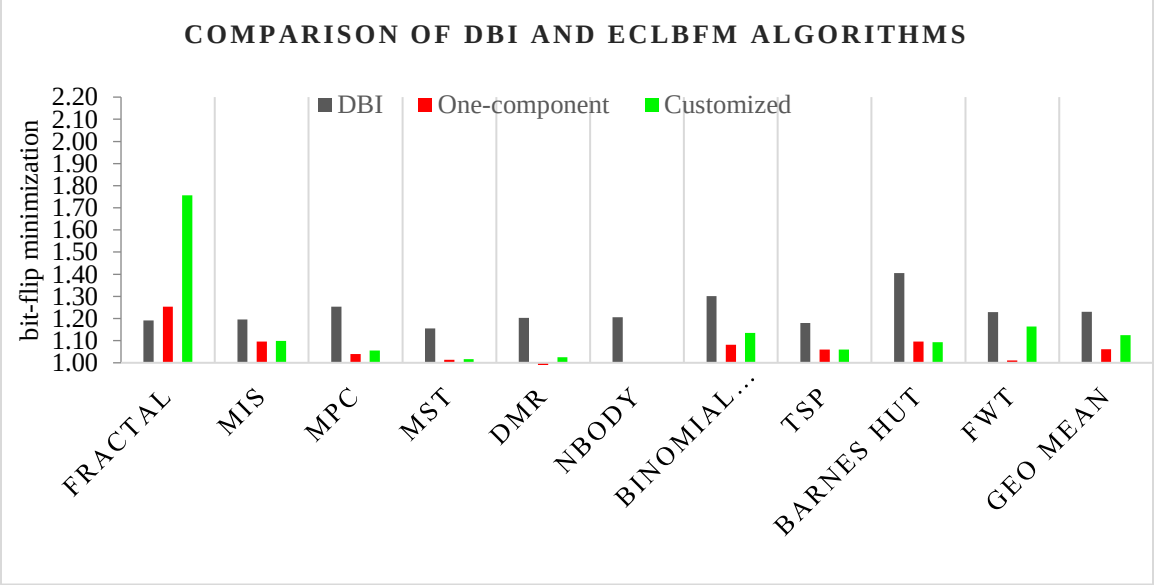


**Figure 9: Comparison between DBI and ECL-BFM one-component algorithm**

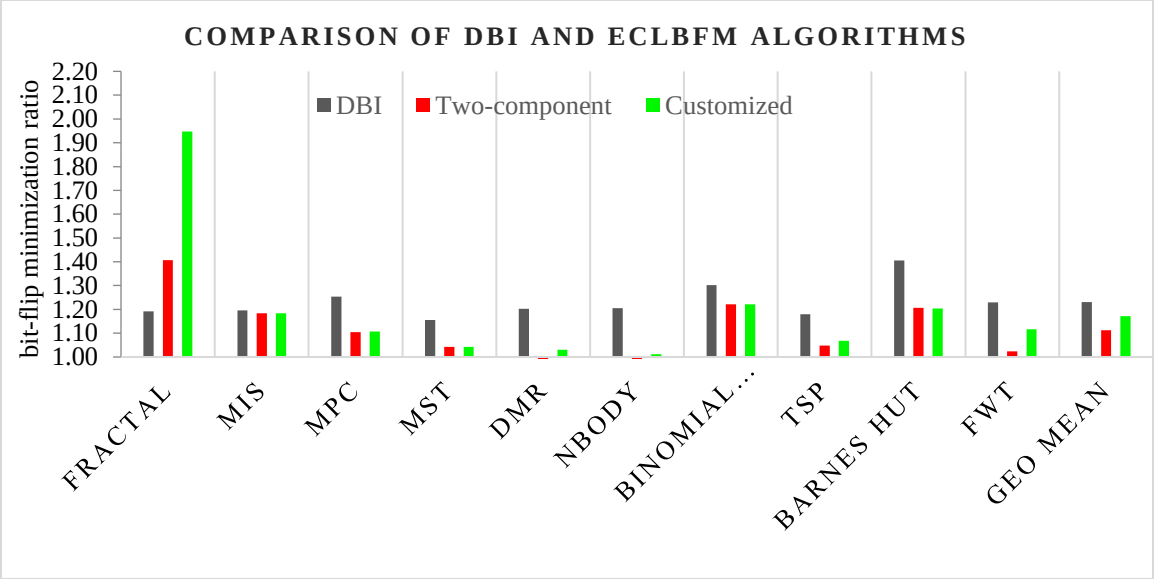


**Figure 10: Comparison between DBI and ECL-BFM two-component algorithm**

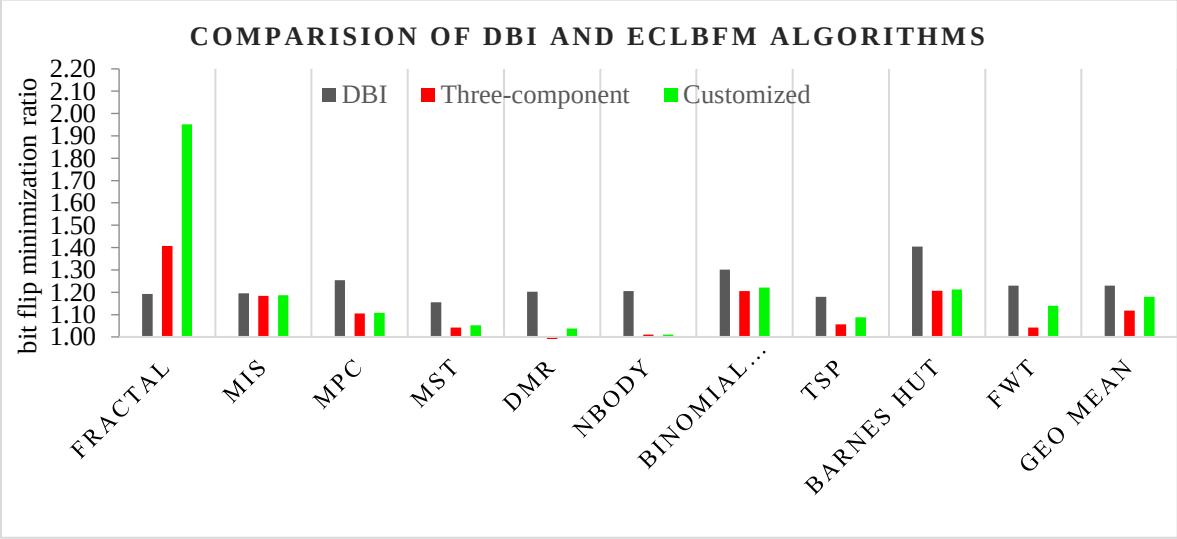


**Figure 11: Comparison between DBI and ECL-BFM three-component algorithm**

# 6. SUMMARY

This thesis introduces new bit-flip minimization algorithms for GPU memory buses. My solution is a chain of simple algorithmic components linked together to form an effective algorithm to reduce bit flips. Every component includes an inverse that performs the opposite transformation to get the original data back.

This thesis proposes two solutions using ECL-BFM approach:

1. Bit-flip minimization algorithms with extra bit line

   This solution improves DBI by introducing a preprocessing step for DBI to reduce more bit flips. It proposes three new algorithms, which reduce 9% more bit flips than DBI. Also, this thesis identifies the customized algorithm for a given input that can on average reduce the bit flips up to 14% over DBI.

2. Bit-flip minimization algorithms without extra bit line

   Adding an extra bit line for DBI is very expensive, therefore ECL-BFM introduces low-cost one-, two-, and three- component algorithms to reduce bit flips without the need for extra bit lines. The new algorithms' performance for some integer inputs reaches that of DBI even without the extra bit line. However, for floating-point inputs, DBI reduces 10% more bit flips.

## 6.1 Future Work

Based on the results from this thesis, future work aims at the following.

1. Identifying more algorithmic components for reducing bit flips.

2. Identifying more input programs and extend the results to include double-precision floats.
3. Extending the algorithmic components to identify an effective minimization algorithm for CPU and other data buses.
4. Implement the algorithm and measure the energy saving in GPUs and other devices.

## REFERENCES


[1] A.Yang, H.Mukka, F. Hesaaraki, and M. Burtscher. MPC: A Massively Parallel Compres-sion Algorithm for Scientific Data. *IEEE Cluster Conference*. September 2015.

[2] Hasmukh P Koringa, Prof. (Dr.) Vipul A Shah and Prof. Durgamadhab Misra. Estimation and Optimization of Power dissipation in CMOS VLSI circuit design.

[3] Mircea R. Stan, Member, IEEE, and Wayne P. Burlesson. Bus-Invert Coding for Low-Power I/O.

[4] Maurizio Skerlj, Paolo Ienne, Ecole Polytechnique Federale de Lausanne (EPFL). Error Protected Data Bus Inversion Using Standard DRAM Components.

[5] Texas Instruments. CMOS Power Consumption and Cpd Calculation. June 1997.

[6] Timothy M. Hollis, Member, IEEE. Data Bus Inversion in High-Speed Memory Applications. IEEE Transactions on Circuits and Systems 4, APRIL 2009.

[7] Stan, Mircea R., and Wayne P. Burleson. "Bus-invert coding for low-power I/O." *IEEE Transactions on very large scale integration (VLSI) systems* 3.1 (1995): 49-58.

[8] Sathish, A., et al. "Energy efficient encoding technique for data-bus in DSM technology." *Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on*. IEEE, 2011.

[9] Stan, Mircea R., and Wayne P. Burleson. "Limited-weight codes for low-power I/O." *International Workshop on low power design*. Vol. 6. No. 3. 1994.

[10] Shin, Youngsoo, Kiyoung Choi, and Young-Hoon Chang. "Narrow bus encoding for low-power DSP systems." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 9.5 (2001): 656-660.

[11] Burtscher, Martin, et al. "Real-time synthesis of compression algorithms for scientific data." *ACM/IEEE International Conference for High-Performance Computing, Networking, Storage and Analysis*. 2016.

[12] https://en.wikipedia.org/wiki/Gray_code