

Journal of Applied Logics

The IfCoLog Journal of Logics and their Applications

Volume 9 • Issue 3 • June 2022

Special Issue: Multiple-Valued Logic

Guest Editors

Martin Lukac

Shinobu Nagayama

Available online at
www.collegepublications.co.uk/journals/ifcolog/
Free open access

JOURNAL OF APPLIED LOGICS - IFCoLoG
JOURNAL OF LOGICS AND THEIR APPLICATIONS

Volume 9, Number 3

June 2022

Disclaimer

Statements of fact and opinion in the articles in Journal of Applied Logics - IfCoLog Journal of Logics and their Applications (JALs-FLAP) are those of the respective authors and contributors and not of the JALs-FLAP. Neither College Publications nor the JALs-FLAP make any representation, express or implied, in respect of the accuracy of the material in this journal and cannot accept any legal responsibility or liability for any errors or omissions that may be made. The reader should make his/her own evaluation as to the appropriateness or otherwise of any experimental technique described.

© Individual authors and College Publications 2022
All rights reserved.

ISBN 978-1-84890-405-7

ISSN (E) 2631-9829

ISSN (P) 2631-9810

College Publications

Scientific Director: Dov Gabbay

Managing Director: Jane Spurr

<http://www.collegepublications.co.uk>

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form, or by any means, electronic, mechanical, photocopying, recording or otherwise without prior permission, in writing, from the publisher.

EDITORIAL BOARD

Editors-in-Chief
Dov M. Gabbay and Jörg Siekmann

Marcello D'Agostino
Natasha Alechina
Sandra Alves
Arnon Avron
Jan Broersen
Martin Caminada
Balder ten Cate
Agata Ciabattoni
Robin Cooper
Luis Farinas del Cerro
Esther David
Didier Dubois
PM Dung
David Fernandez Duque
Jan van Eijck
Marcelo Falappa
Amy Felty
Eduaro Fermé

Melvin Fitting
Michael Gabbay
Murdoch Gabbay
Thomas F. Gordon
Wesley H. Holliday
Sara Kalvala
Shalom Lappin
Beishui Liao
David Makinson
Réka Markovich
George Metcalfe
Claudia Nalon
Valeria de Paiva
Jeff Paris
David Pearce
Pavlos Peppas
Brigitte Pientka
Elaine Pimentel

Henri Prade
David Pym
Ruy de Queiroz
Ram Ramanujam
Chrtian Retoré
Ulrike Sattler
Jörg Siekmann
Marija Slavkovik
Jane Spurr
Kaile Su
Leon van der Torre
Yde Venema
Rineke Verbrugge
Heinrich Wansing
Jef Wijsen
John Woods
Michael Wooldridge
Anna Zamansky

SCOPE AND SUBMISSIONS

This journal considers submission in all areas of pure and applied logic, including:

pure logical systems	dynamic logic
proof theory	quantum logic
constructive logic	algebraic logic
categorical logic	logic and cognition
modal and temporal logic	probabilistic logic
model theory	logic and networks
recursion theory	neuro-logical systems
type theory	complexity
nominal theory	argumentation theory
nonclassical logics	logic and computation
nonmonotonic logic	logic and language
numerical and uncertainty reasoning	logic engineering
logic and AI	knowledge-based systems
foundations of logic programming	automated reasoning
belief change/revision	knowledge representation
systems of knowledge and belief	logic in hardware and VLSI
logics and semantics of programming	natural language
specification and verification	concurrent computation
agent theory	planning
databases	

This journal will also consider papers on the application of logic in other subject areas: philosophy, cognitive science, physics etc. provided they have some formal content.

Submissions should be sent to Jane Spurr (jane@janespurr.net) as a pdf file, preferably compiled in \LaTeX using the IFCoLog class file.

CONTENTS

ARTICLES

- Editorial Note for the Special Issue on Multiple-Valued Logic 651**
Martin Lukac and Shinbo Nagayama
- Implementation of CMOS Invertible Logic on Zynq-SoC Platform:
A Case Study of Training BNN 653**
Duckgyu Shin, Naoya Onizawa and Takahiro Hanyu
- Efficient PAM-4 Symbol Estimation Using Soft Clustering 675**
Yosuke Iijima and Yasushi Yuminaka
- Data-Classification-Based Determination for Ophthalmological
Examination Categories using Machine Learning 691**
Shoji Morita, Teujiro Isokawa, Naotake Kamiura and Hitoshi Tabuchi
- Construction Algorithms for Bent Functions Derived
from their Particular Properties 711**
Radomir Stanković, Milena Stanković, Claudio Moraga and Jaakko Astola
- Quantum Machine Learning, Logic Minimization, and Circuit Design
by Optimizing Ternary-Input Binary-Output Kronecker
Reed-Muller Forms 733**
Maggie Bao, Cole Powers and Marek Perkowski

DNA Technology for Multi-Valued Data Storage using	
Read Only Memory	781
<i>Hafiz Md. Hasan Babu, Khandaker Mohammad Mohi Uddin, Tamanna Tabassum</i>	
<i>and Mohammed Nasir Uddin</i>	
Ternary Functions with Bent Reed-Muller-Fourier Spectra	805
<i>Claudio Moraga, Radomir Stanković and Milena Stanković</i>	

EDITORIAL NOTE TO THE SPECIAL ISSUE ON MULTIPLE-VALUED LOGIC

The year 2022 seems to finally bring a glimpse of post-COVID-19 life with hope to return to a less isolated situation. While the COVID-19 was winding down, quantum computing has been advancing in certainly giant steps. Between solid state quantum computers to processors of up to thousands of qubits, the next generation beyond NISQ devices seems to be behind the door. Multiple valued quantum qubits have also been shown of great use and future quantum computers are very likely to use for specific purpose more than two values. In parallel to quantum computers other emerging technologies have been advancing in particular when related to the machine learning applications, general purpose quantum computing and several technologies related o high speed parallel massive data processing. Therefore this year's special issue on Multiple-Valued Logic includes papers from various areas of logic, logic design, circuits and emerging technologies.

The first paper entitled *Implementation of CMOS invertible logic on Zynq-SoC Platform : A Case Study of Training BNN* discusses the usage and implementation of binarized neural networks on a FPGA simulator. In particular the article addresses an invertible implementation: an alternative probabilistic computational model allowing to provide bi-directional operations using stochastic computing. The paper focuses on accelerating the PC to FPGA transfer latency by proposing to transfer the data by using the AXI interface. The results show a considerable improvement.

The second paper entitled *Efficient PAM-4 Symbol Estimation Using Soft Clustering* introduces multi-valued data transmission using four-level pulse amplitude modulation (PAM-4) for the reduction bandwidth limitation in interconnects. The quality of the transmitted signal is measured using an eye-opening monitor that uses statistical properties of PAM-4. The results and simulations show the feasibility of adaptive equalization for PAM-4 signaling.

The third paper entitled *Data-Classification-Based Determination for Ophtalmological Examination Categories Using Machine Learning* discusses a method for determining examination categories using machine learning. The data classified are patient hand-filled questionnaires. The paper is concerned with in particular the

comparison of combination of word splitting approaches, word representation and classification methods. As a result the paper shows that for this particular problem the best combination is Sudachi, One Hot encoding and CatBoost.

The fourth paper entitled *Construction Algorithms for Ternary Bent Functions Derived from Their Particular Properties* studies the algorithmic construction of ternary bent functions. The proposed method is based on the information contained in the value vector of bent functions resulting in six classes of ternary bent function. To design bent functions within a class by permutation matrices with a specific block structure. As a consequence, bent functions can be transformed to matrix and vector representation of smaller length and therefore allow a faster generation of new bent functions by permutation matrices on these shorter representations.

The fifth paper entitled *Quantum Machine Learning, Logic Minimization, and Circuit Design By Optimizing Ternary-Input Binary-Output Kronecker Reed-Muller Forms* the authors propose new spectral transform for ternary-input binary-output functions that generalizes the binary Kronecker Reed-Muller forms. The authors generalize past quantum Grover-based algorithms presented for the binary Fixed-Polarity Reed-Muller and Kronecker Reed-Muller transforms. The algorithm can be applied to incompletely specified functions, thus, introducing a new approach to quantum Machine Learning.

In the next paper entitled *DNA Technology for Multi-Valued Data Storage Using Read Only Memory* the authors present a multiple valued ROM architecture based on the principles of DNA based computers. The high parallelism and the higher radix of the DNA computing system allows to increase storage capacity.

The last paper entitled *Ternary Functions with Bent Reed-Muller-Fourier Spectra* studies ternary functions which have a bent Reed-Muller-Fourier spectrum. The studied 2-place functions are described in six classes and are shown to be related by spectral invariance operations. Such ternary bent functions that have the same value vector as their RMF bent spectra known as fixed points are generalized in this paper to rotational fixed points. As a result a method to generate n -place ternary functions with bent RMF spectrum when $n > 2$ is proposed.

I hope that you will find this year's special issue exciting and motivating your next advances in the exciting area of Multiple-Valued Logic and emerging Technologies

Martin Lukac
Nazarbayev University
Kazakhstan

Shinobu Nagayama
Hiroshima City University
Japan

Guest Editors

IMPLEMENTATION OF CMOS INVERTIBLE LOGIC ON ZYNQ-SOC PLATFORM: A CASE STUDY OF TRAINING BNN

DUCKGYU SHIN

School of Engineering, Tohoku University, Japan

Research Institute of Electrical Communication, Tohoku University, Japan

`duckgyu.shin.p4@dc.tohoku.ac.jp`

NAOYA ONIZAWA

Research Institute of Electrical Communication, Tohoku University, Japan

`naoya.onizawa.a7@tohoku.ac.jp`

TAKAHIRO HANYU

Research Institute of Electrical Communication, Tohoku University, Japan

`takahiro.hanyu.c4@dttohoku.ac.jp`

Abstract

In this article, we introduce a prototype of a Zynq system-on-chip (SoC)-based hardware platform for complementary metal-oxide-semiconductor (CMOS) invertible logic. CMOS invertible logic realizes probabilistic bidirectional operations (forward and backward) using stochastic computing. Using this unique feature, CMOS invertible logic hardware for several challenging problems, such as factorization and the training of neural networks, can be implemented on field-programmable gate arrays (FPGAs). However, a problem with conventional FPGA-based implementations is the latency of data transfer between the PC and the FPGA because of the universal asynchronous receiver-transmitter (UART) interface, which can transmit only one byte of information at once. The SoC-based CMOS invertible hardware proposed in this work is implemented on a Xilinx Zynq-7000, and it communicates with the embedded CPU (ARM Cortex-A9) via an Advanced eXtensible Interface (AXI); thus, the data transfer latency is much less than that of a UART interface. For performance evaluation, the proposed SoC-based training hardware was used to train a 2-layer binarized convolutional neural network (BCNN) model using a simplified dataset based on the Modified National Institute of Standards and Technology (MNIST) database as the training dataset, and the results were compared with those of a conventional FPGA implementation. The proposed implementation achieved 14.4x lower data transfer latency.

1 Introduction

Invertible logic provides the ability to perform probabilistic bidirectional operations (in both forward and backward modes), which are difficult to realize in typical binary logic gates [1]. In forward mode, an invertible logic circuit produces outputs corresponding to given inputs, similar to a typical binary logic circuit. In contrast, in backward mode, the circuit probabilistically determines the inputs that correspond to a given output. This unique feature is derived from a Boltzmann machine [2] and a probabilistic nanomagnetic device model [3]. The capability of bidirectional operations is realized by driving the energy of a network to converge to the global minimum under noise signals.

However, invertible logic as described in [1] is difficult to implement in hardware because the device model is designed using a magnetic tunnel junction [4]. In complementary metal–oxide–semiconductor (CMOS) invertible logic [5], the device model is approximated by means of stochastic computing [6]. Through such approximation, invertible logic circuits can be implemented on field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs). Thus, with CMOS invertible logic, hardware solutions can be easily implemented for several challenging problems, such as integer factorization [5] (invertible multipliers), combinational optimization, and the training of binarized convolutional neural networks (BCNNs) [7, 8]. For example, training hardware based on CMOS invertible logic [9, 8] can achieve a training process latency of 0.067 s, which is approximately 40x faster than training on a CPU. However, the total latency including data transfer is 1.92 s; hence, data transfer represents a major bottleneck. Conventional CMOS invertible logic circuits are implemented on FPGAs, and such a circuit sends and receives data using a universal asynchronous receiver-transmitter (UART) interface. A UART interface can transmit only one byte of information at once; thus, the latency of data transfer constitutes most of the latency of the entire process for conventional CMOS invertible logic hardware. These FPGA-based implementations are not suitable for applications that require the processing of enormous amounts of data, such as the training of a BCNN.

In this paper, we present the design flow of a system-on-chip (SoC)-based CMOS invertible logic platform for efficient data transfer. A Xilinx Zynq SoC FPGA board has an embedded processor and uses the Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) specification for communication between a programmable logic (PL) circuit and the embedded processor (PS). By using an AXI interface, such a SoC-based CMOS invertible logic circuit can realize faster data transfer than an FPGA-based implementation. As a case study, training hardware for a 2-layer BCNN model was implemented on a Xilinx Zynq-7000 chip

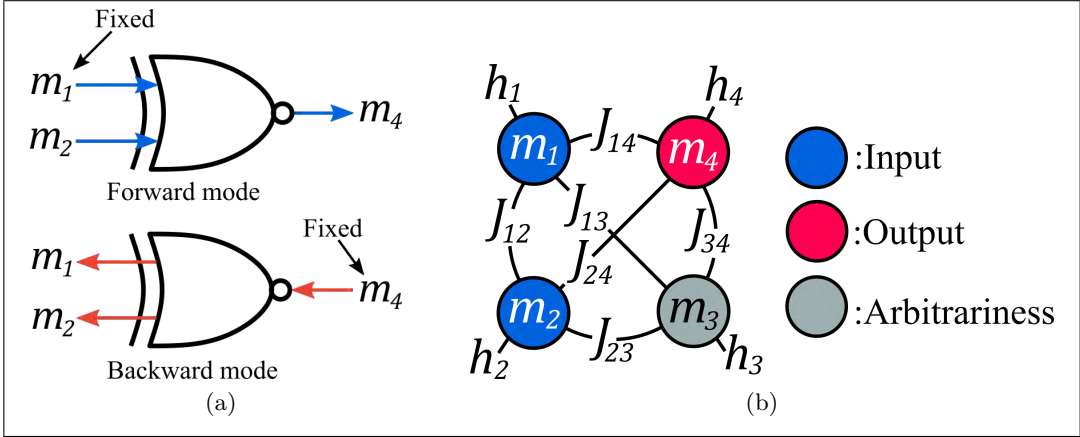


Figure 1: (a) Example of an invertible logic XNOR gate. In forward mode, the output (m_4) is calculated from the given inputs (m_1 and m_2). In backward mode, the inputs corresponding to a fixed output are obtained. (b) The Hamiltonian configuration of an invertible XNOR gate. m_1 and m_2 correspond to the inputs to the XNOR gate, and m_4 corresponds to the output. m_3 is arbitrary.

and used to train a BCNN model using a simplified dataset based on the Modified National Institute of Standards and Technology (MNIST) database [10]. The data transfer latency of the proposed SoC-based training hardware is 0.128 s, which is approximately 14.4x lower than that of FPGA-based training hardware for the same amount of data [8].

The rest of this paper is organized as follows. Section 2 reviews CMOS invertible logic and its applications. Section 3 describes the design flow of the prototype SoC-based CMOS invertible logic circuit for BCNN training hardware. Section 4 evaluates the cognition accuracy and the training speed and compares them with those of FPGA-based training hardware based on CMOS invertible logic. Section 5 concludes the paper.

2 Preliminaries

2.1 CMOS invertible logic

CMOS invertible logic enables CMOS digital circuits to perform probabilistic bidirectional operations [5]. Fig. 1 (a) illustrates an example of bidirectional operations using an invertible XNOR gate. In forward mode, the inputs (m_1 and m_2) are fixed and an output (m_4) is computed, similar to a typical binary logic gate. In backward

Truth table				Node states				E	Validity
m_1	m_2	m_3	m_4	m_1	m_2	m_3	m_4		
0	0	0	0	-1	-1	-1	-1	-2	Invalid
1	0	0	0	+1	-1	-1	-1	4	Invalid
0	1	0	0	-1	+1	-1	-1	4	Invalid
0	0	1	0	-1	-1	+1	-1	-2	Invalid
0	0	0	1	-1	-1	-1	+1	-4	Valid
1	1	0	0	+1	+1	-1	-1	14	Invalid
1	0	1	0	+1	-1	+1	-1	-4	Valid
1	0	0	1	+1	-1	-1	+1	-2	Invalid
0	1	1	0	-1	+1	+1	-1	-4	Valid
0	1	0	1	-1	+1	-1	+1	-2	Invalid
0	0	1	1	-1	-1	+1	+1	4	Invalid
1	1	1	0	+1	+1	+1	-1	-2	Invalid
1	1	0	1	+1	+1	-1	+1	4	Invalid
1	0	1	1	+1	-1	+1	+1	-2	Invalid
0	1	1	1	-1	+1	+1	+1	-2	Invalid
1	1	1	1	+1	+1	+1	+1	-4	Valid

Table 1: The truth table of an XNOR gate and the table of the node states. The gate has 4 nodes; thus, there are 2^4 possible state combinations in total. When every state corresponds to the correct behavior of an XNOR gate, h and J are determined such that the energy is minimal. -1 means a logical value of ‘0’, and +1 means ‘1’. The minimal energy is -4 in the case of an XNOR gate.

mode, the output (m_4) is fixed, and the possible combinations of inputs corresponding to a fixed output are probabilistically determined. For instance, if m_4 is fixed to ‘1’, the inputs are determined to be $(m_1, m_2) = (1, 0)$ or $(0, 1)$, which are the possible input combinations for an XNOR gate. A CMOS invertible logic circuit is a network of nodes based on a Boltzmann machine [2]. A network configuration for an invertible XNOR gate, as shown in Fig. 1 (b), consists of four nodes categorized as input nodes (m_1 and m_2), an output node (m_4), and an arbitrary node (m_3). Each node has a bias (h_i), and it is connected to each other node with corresponding interaction weights (J_{ij}). The values of h_i and J_{ij} are determined by the Hamiltonian H , which is the energy of the network and is defined as follows:

$$H = - \sum_i h_i m_i - \sum_{i < j} J_{ij} m_i m_j, \quad (1)$$

where $m_i \in [-1, 1]$ represents the output of a node (a node state). In Eq. (1), m_i can take values of -1 and +1, where -1 corresponds to a logical value of ‘0’ and +1 corresponds to a logical value of ‘1’. The Hamiltonian of a basic logic gate, such as an XNOR gate or an AND gate, is obtained by ground-state spin logic [11, 12]. In ground-state spin logic, the energy is at the global minimum when all nodes are in valid states for the function that is to be computed via invertible logic. For example, $[m_1, m_2, m_3, m_4] = [0, 0, 0, 1]$ are valid states for an embedded XNOR gate; thus, the energy is at the global minimum. In contrast, the energy is greater than the global minimum when some nodes are not in valid states. The Hamiltonian of an XNOR gate involves four nodes, as illustrated in Fig. 1 (b), although the XNOR gate itself has only two inputs and one output. This is because a Hamiltonian that contains only three nodes cannot represent the global minimum when all nodes are in valid states for an XNOR gate [12]. With the addition of an arbitrary node, the Hamiltonian becomes able to represent the global minimum for an XNOR gate. Table 1 shows the truth table for an XNOR gate and the node states in the Hamiltonian of the XNOR gate. Based on ground-state spin logic, h_i and J_{ij} of the invertible XNOR gate are determined as follows:

$$m_i = \begin{bmatrix} m_1 & m_2 & m_3 & m_4 \end{bmatrix},$$

$$h_i = \begin{bmatrix} -1 & -1 & +2 & +1 \end{bmatrix}, \quad (2a)$$

$$J_{ij} = \begin{bmatrix} 0 & -1 & +2 & +1 \\ -1 & 0 & +2 & +1 \\ +2 & +2 & 0 & -2 \\ +1 & +1 & -2 & 0 \end{bmatrix}. \quad (2b)$$

Fig. 2 shows an example of the landscape of a Hamiltonian. The Hamiltonian is designed to reach its global minimum when all nodes are in valid states for the desired function. Therefore, bidirectional operation of a CMOS invertible logic circuit is realized by converging the Hamiltonian (energy) to the global minimum. When a CMOS invertible logic circuit is operating, the energy is driven to converge by fluctuations in the unfixed nodes. For instance, when an invertible XNOR gate operates in backward mode, the output node (m_4) is fixed, while the input and arbitrary nodes (m_1, m_2 , and m_3) fluctuate. The unfixed nodes are driven to fluctuate using noise signals (n_{rnd}) to prevent the energy from becoming trapped in a local minimum. The details of these noise signals are introduced in the next subsection.

In CMOS invertible logic, the probabilistic behavior of the nodes can be approximated using CMOS digital circuits by means of stochastic computing [6]; thus, such

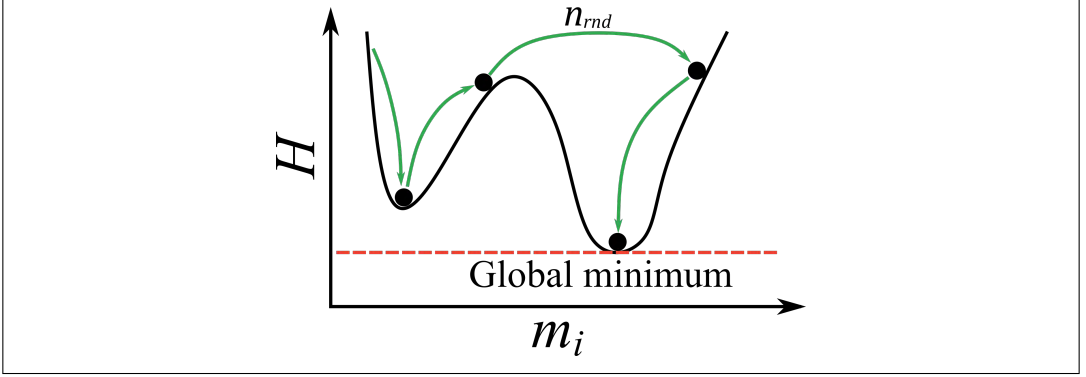


Figure 2: The energy convergence of the Hamiltonian. In invertible logic, the energy is driven by noise signals (n_{rnd}) to converge to the global minimum, when all nodes take valid states.

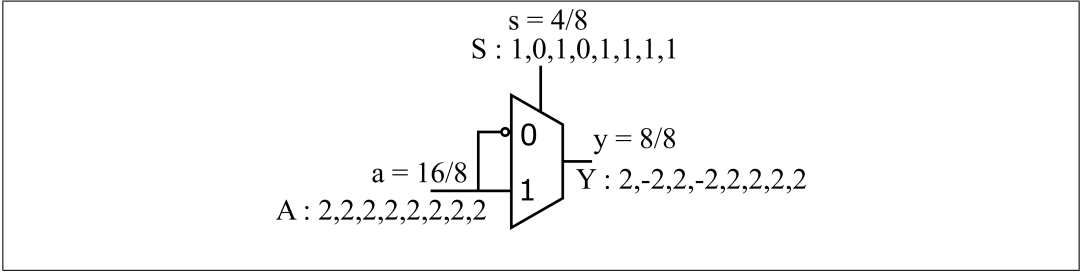


Figure 3: The multiplication of binary and integer stochastic bit streams ($y = a \cdot s$). Multiplication in stochastic computing can be realized using a multiplexer.

circuits can be easily implemented on FPGAs or ASICs [5]. In stochastic computing, values are represented by frequencies of ‘1’s in random bit streams. Let us denote by $x \in \{0, 1\}$ a stochastic bit stream, and let the probability of the appearance of a ‘1’ in this bit stream be denoted by P_x . In binary stochastic computing, in which the range of the represented values X is -1 to +1, X is defined as $X = 2 \cdot P_x - 1$. On the other hand, integer stochastic computing requires the representation of real values $X \in [-r, +r]$, where $r \in \{1, 2, \dots\}$, using several stochastic bit streams [13]. Stochastic computing has recently been used in several applications to achieve area-efficient hardware implementations [14, 15]. The behavior of a node is defined using stochastic computing as follows:

$$m_i(t + \tau) \approx \text{sgn}(\tanh(I_i(t + \tau))), \quad (3a)$$

$$I_i(t + \tau) \approx h_i + \sum J_{ij} m_j(t) + n_{rnd} \cdot \text{sgn}(\text{rnd}(-1, +1)), \quad (3b)$$

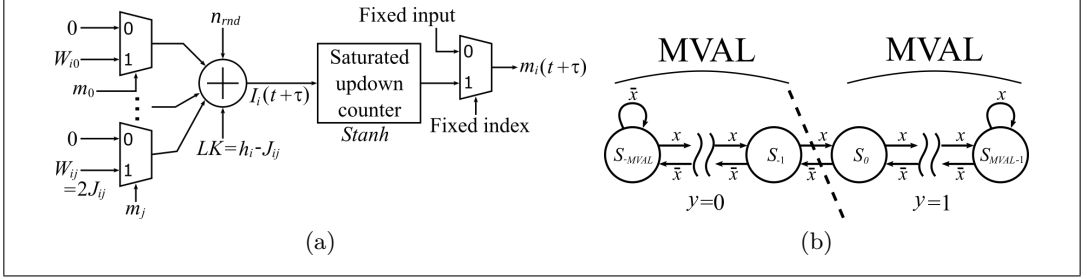


Figure 4: (a) Block diagram of a spin gate. In CMOS invertible logic, the nodes are implemented using CMOS digital circuits called ‘spin gates’. (b) The FSM of the saturated bit counter in the spin gate. The tanh function is approximated to Stanh, which is implemented in the FSM. The number of states is determined by the noise parameter MVAL.

where n_{rnd} is a parameterized noise signal and t and τ are cycle times. The tanh function in Eq. (3a) is approximated to Stanh (the stochastic tanh function) using stochastic computing [15]. The details of the approximation in stochastic computing are described in the next subsection. Fig. 3 shows the multiplication of a binary stochastic bit stream and an integer stochastic bit stream using a multiplexer. The output of a node, $m_i(t)$, is a binary stochastic bit stream; therefore, the multiplications of J_{ij} and $m_j(t)$ can be computed by multiplexers [6, 14].

2.2 Hardware implementation based on CMOS invertible logic

The probabilistic node operations defined in Eq. (3a) and Eq. (3b) are implemented by a ‘spin gate’ using a CMOS digital circuit [5]. Fig. 4 (a) shows a block diagram of a spin gate. The output of the spin gate, m_i , is either “-1” or “+1” in Eq. (3a) and Eq. (3b); however, m_i is represented as taking logical values of ‘0’ or ‘1’. Therefore, h_i and J_{ij} are modified to be consistent with a spiking neuron model, and LK_i and W_{ij} are given as follows:

$$LK_i = h_i - \sum_j J_{ij}, \quad (4a)$$

$$W_{ij} = J_{ij}. \quad (4b)$$

Using this spiking neuron model, multiplications and summations of the interaction weights can be implemented using multiplexers. The tanh function is approximated to Stanh, which is realized as a finite state machine (FSM), as shown in Fig. 4 (b). The Stanh function is given as follows:

$$\text{Stanh}(\text{MVAL}, x) \approx \tanh(x \cdot \text{MVAL}/2). \quad (5)$$

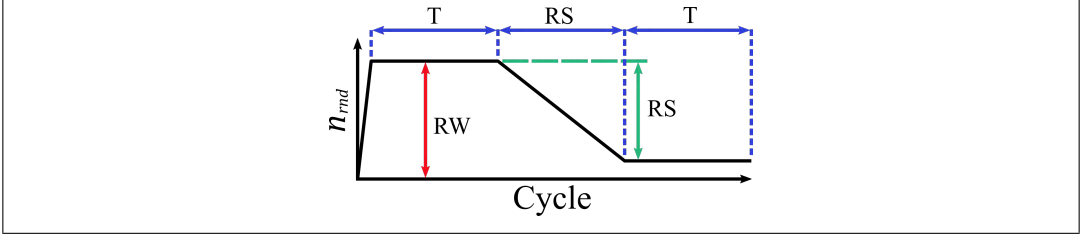


Figure 5: Example of a parameterized noise signal. The noise signal is controlled by 5 noise parameters, namely, RW, RS, T, MVAL, and α .

The number of FSM states is determined to be $2 \cdot \text{MVAL}$, where MVAL is a noise parameter. The spin gate computes its output m_i when the output is not fixed; otherwise, it delivers only a fixed input value.

The energy of a CMOS invertible logic circuit is driven to convergence using parameterized noise signals n_{rnd} . Fig. 5 shows an example of such a noise signal, including the noise parameters. The parameterized noise signal is controlled by five parameters, namely, RW, T, RS, MVAL, and α . RW is the magnitude of the noise signal, and T is the cycle time for which the noise magnitude remains constant. RS represents a reduction in the noise signal; the noise signal decreases by RS over RS cycles. If RS is 0, the magnitude of the noise signal remains constant over $2T$ cycles. α is a scaling factor of the noise signal; the noise signal scaled by α is given as follows:

$$n'_{rnd} = 2^\alpha \cdot n_{rnd}. \quad (6)$$

Such a scaled noise signal is applied to a node that needs stronger noise than other nodes. As mentioned before, MVAL is a parameter of the FSM shown in Fig. 4 (b).

CMOS invertible logic has recently been adopted in several applications, such as an invertible multiplier (factorization) [5] and training hardware for BCNNs [7, 8]. These hardware designs were implemented on ASICs or FPGAs, and they achieved fast operation times and high power efficiency. For example, in our previous work, training hardware for a 2-layer BCNN model achieved a training time of 0.067 s when using 100 training samples. However, the entire training time, including the latency of data transfer, was 1.92 s, of which the hardware operation time accounted for only 3.5%. This training hardware for BCNNs was implemented on an FPGA, and it received the noise parameters and the training dataset through a UART interface [16]. A UART data frame is only 8 bits in length, and this is not suitable for training neural networks because such training requires enormous amounts of data.

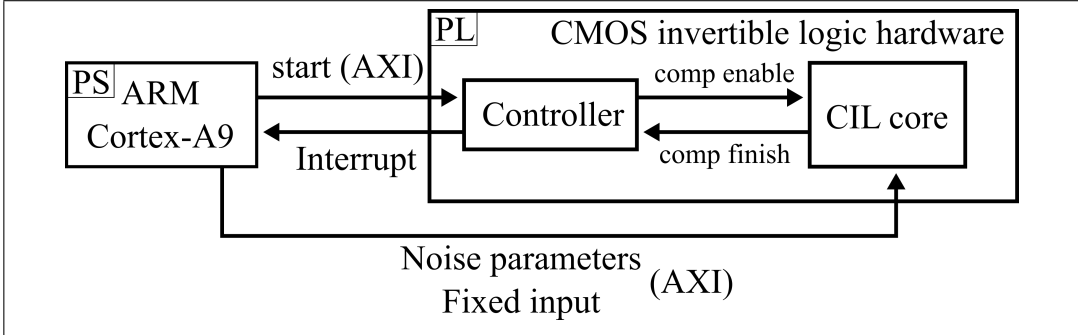


Figure 6: Block diagram of the SoC-based CIL hardware prototype. The programmable logic (PL) part contains a CIL core, which is designed based on the Hamiltonian; a controller for the CIL core; and AXI interfaces. The embedded ARM Cortex-A9 (PS) transmits the fixed input and noise parameters through AXI interfaces. The CIL core informs the processor through an interrupt signal that the operation for the given fixed input has been completed.

3 Design flow of the SoC-based invertible logic prototype

Fig. 6 shows the architecture of our SoC-based CMOS invertible logic (CIL) hardware prototype. The PL part contains a CIL core, a controller for the CIL core, and an AXI slave interface to communicate with the embedded processing system (PS). When the hardware operates, the CIL core receives the noise parameters and the fixed input data from the processor via the AXI interface, and then, the controller initiates the bidirectional operation of the CIL core by means of a comp enable signal. If a correct output for the given input is obtained, the CIL core notifies the controller that the operation has been completed by means of a comp finish signal, and then, the controller informs the processor using an interrupt signal. Finally, the output obtained from the bidirectional operation is transmitted by the AXI interface to the processor. The prototype hardware is composed of the PS part and the PL part; thus, each part of the hardware requires a proper design flow.

3.1 PL design flow

The design flow of the PL part of the prototype is shown in Fig. 7 (a). The CIL core shown in Fig. 6 is designed based on the Hamiltonian converted from the function that is to be computed via CIL. However, it is difficult to obtain a Hamiltonian with a complex structure, such as the Hamiltonian for training hardware, using

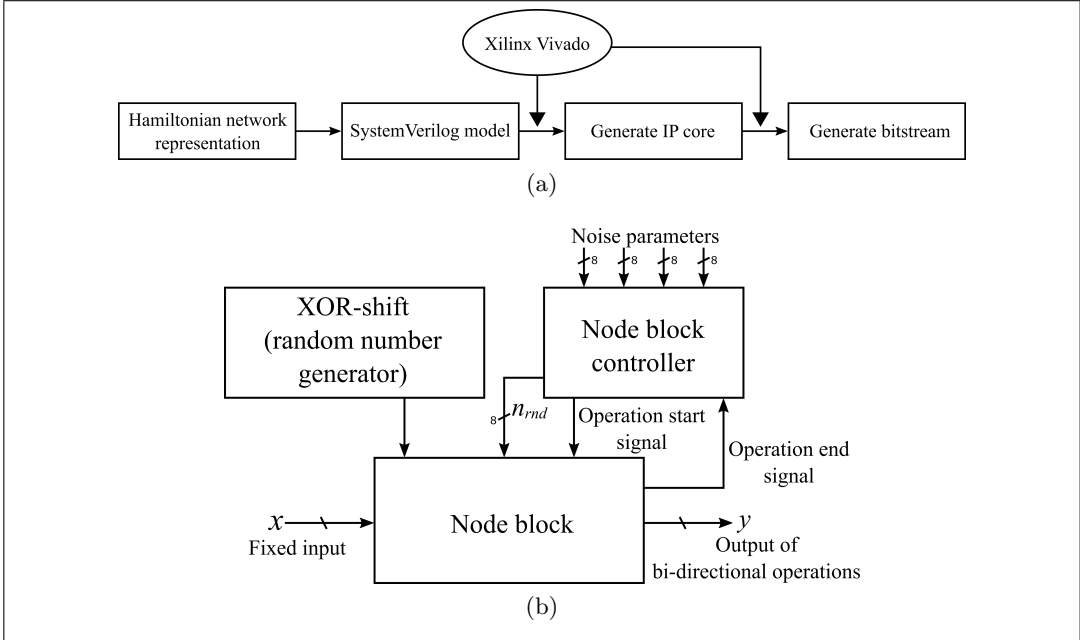


Figure 7: (a) Design flow of the PL part. The CIL hardware is designed based on the Hamiltonian converted from a desired function. (b) Block diagram of the CIL core. The node block is designed based on the Hamiltonian. The XOR-shift generator generates random noise signals, and the node block controller controls the magnitude of the noise signals and the operation of the node block.

ground-state spin logic. Therefore, the final complex Hamiltonian is obtained by combining smaller Hamiltonians, such as the Hamiltonians of individual logic gates [17]. Fig. 8 (a) and (b) illustrate the process of combining Hamiltonians. The example circuit contains two XNOR gates (XNOR1 and XNOR2), and the output of XNOR1 (m_4) is used as an input to XNOR2. The Hamiltonian of each XNOR gate can be represented as a matrix, as shown in Fig. 8 (b). m_4 is a common node in both Hamiltonians, and the Hamiltonian of the entire example circuit can be represented as a single matrix, as shown in Fig. 8 (b). Additionally, the biases are combined by summing the biases of the common nodes. The network representation of the Hamiltonian is converted into a SystemVerilog model. The CIL core consists of a node block, an XOR-shift generator [18], and a node block controller, and its block diagram is shown in Fig. 7 (b). The node block, which is a main component of the CIL core, consists of spin gates (p-bits) in accordance with the network structure of the Hamiltonian. The Hamiltonian is represented as a vector and the weight map

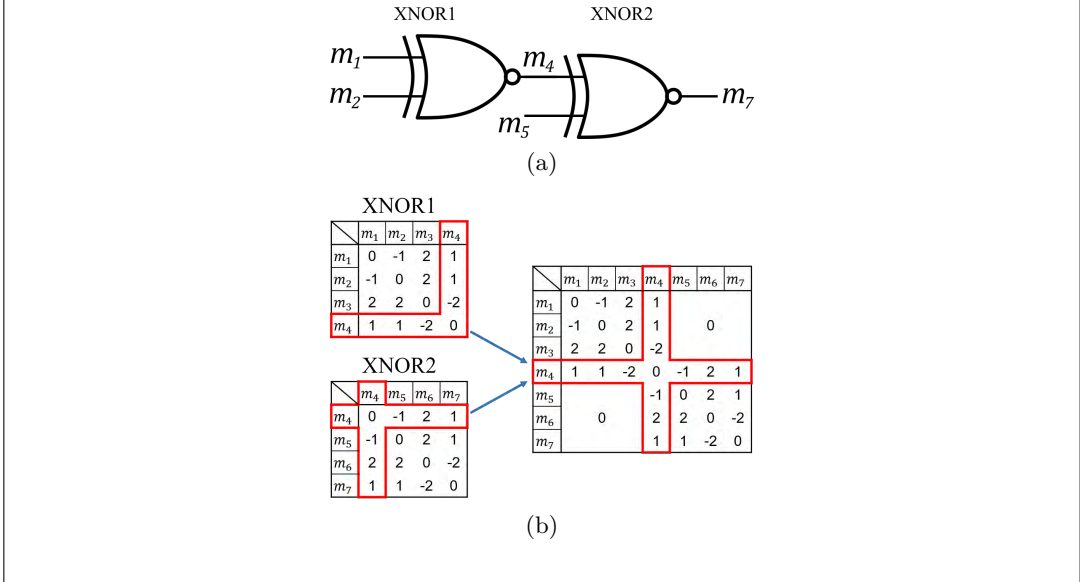


Figure 8: Example of the combination of Hamiltonians. (a) A logic circuit diagram of two XNOR gates (XNOR1 and XNOR2). The output (m_4) of XNOR1 is used as an input to XNOR2. (b) The combination of the Hamiltonians of the two XNOR gates. m_3 and m_6 are arbitrary nodes, m_1 and m_2 are the input nodes of XNOR1, and m_4 is its output node. m_4 is used as an input to XNOR2.

for assignment to the spin gates in the Verilog design. The Verilog snippet for the example circuit shown in Fig. 8 (a) is given below:

```

module node_block
#(parameter integer WEIGHT_WIDTH = ...); // Parameter declaration.
(output_signals, input_signals, ... ); // I/O declaration.
// Number of nodes.
parameter integer N = 7;
parameter integer NUM_WEIGHTS = 12;
parameter integer NS [0:N-1] = '{3,3,3,6,3,3,3};
// Biases.
parameter integer LEAKS [0:N-1] = '{1,1,-8,2,1,-8,1};
// Interaction weights.
parameter integer WEIGHTS [0:NUM_WEIGHTS-1] = '{-2,4,-2,4,-2,4,-2,4,-2,4,-2,4};
parameter integer WEIGHT_MAP [0:N-1] [0:6-1] =
    '{0,1,2,-1,-1,-1},
    '{0,3,4,-1,-1,-1},
    '{1,3,5,-1,-1,-1},
    '{2,4,5,6,7,8},
    '{6,9,10,-1,-1,-1},
    '{7,9,11,-1,-1,-1},
    '{8,10,11,-1,-1,-1};
parameter integer OUT_MAP [0:N-1] [0:6-1] =
    '{1,2,3,-1,-1,-1},
    '{0,2,3,-1,-1,-1},
    '{0,1,3,-1,-1,-1},
    '{0,1,2,4,5,6},
    '{3,5,6,-1,-1,-1},
    '{3,4,6,-1,-1,-1},

```

```

    '{3,4,5,-1,-1,-1}';
// ...
for (i=0; i<N; i++) begin
    wire [NS[i]-1:0] pbit_input_signals;
    wire signed [WEIGHT_WIDTH-1:0] pbit_weight_vector [0:NS[i]-1];

    for (j=0; j<NS[i]; j++) begin
        assign pbit_weight_vector[j] = pbit_weight_dout[WEIGHT_MAP[i][j]];
        assign pbit_input_signals[j] = pbit_output_signals[OUT_MAP[i][j]];
    end
    // Assign interaction weights to p-bit using WEIGHTS and WEIGHT_MAP.
    pbit #(pbit_parameters) pbit_inst (.weights(pbit_weight_vector));
    assign output_signals[i] = pbit_output_signals[i];
end
endmodule

```

The XOR-shift generator generates random numbers for the noise signals, and the node block controller controls the operations of the node block, including controlling the noise signals. Following the design of the CIL core, a custom IP core based on the CIL core is generated through FPGA design tools, such as Xilinx Vivado, for hardware/software codesign. The Verilog snippet for the CIL core is given below:

```

module CIL_core
    #(parameter integer WEIGHT_WIDTH = ...); // Parameter declaration.
    (output_signals, input_signals, ... ); // I/O declaration.

    // AXI interface declaration from the Xilinx vivado AXI example.

    node_block node_inst (...) // Node block instance
    ctr_mult ctr_inst (...) // Node block controller instance
    xorshift prng_inst (...) // XOR shift instance

endmodule

```

By means of this IP core, the CIL core can be merged easily with the IP core of the Zynq processor or the AXI interface. Finally, the bitstream file for implementation is generated from the entire PL design, comprising the IP cores for the embedded processor and the CIL core.

3.2 PS design flow

Fig. 9 (a) shows the design flow of the proposed SoC-based CIL hardware. The operating system (OS) of the prototype hardware is Linux, and Das U-Boot is used as the bootloader. While booting the Linux system, a device tree [19] file is required for Linux to correctly recognize the CIL core, and it is generated by the Xilinx Hardware Software Interface (HSI) tool. Additionally, to utilize the custom IP in the system, a device driver is needed. In general, a device driver runs in the kernel space of a Linux system; however, the driver of the CIL core is run in the userspace through a Universal I/O (UIO) interface [20]. The UIO framework enables the development of drivers similar to those of software applications. Thus, it is easy to adapt to custom devices such as the CIL core. Additionally, the SoC-based CIL hardware introduced in this paper is a prototype, so a UIO driver is suitable for further improvements. The UIO driver allows access directly to the registers of

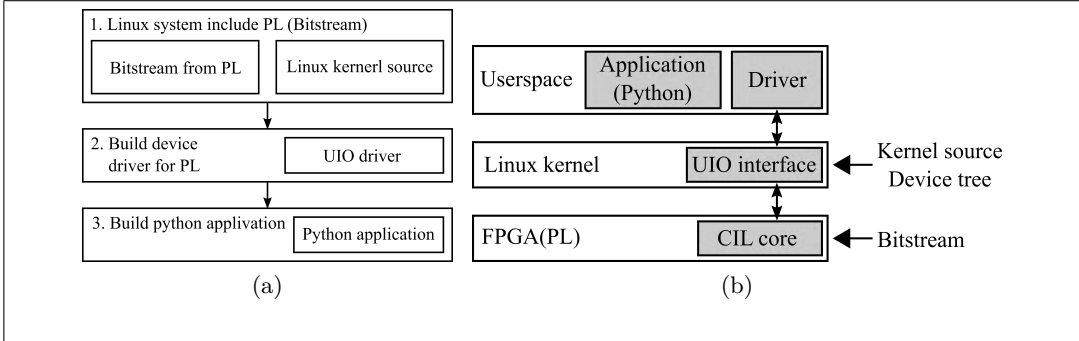


Figure 9: (a) The design flow of the PS. The OS is Linux, including the bitstream from the PL part. After the OS is built, the device driver is built using a UIO driver. Finally, a Python application is prepared to use the CIL hardware. (b) The structure of the PS including the PL part. In general, device drivers run in a kernel space; however, the driver for the CIL core, which is a custom IP, runs in the userspace through the UIO interface.

the hardware and device files through a device file located in ‘/dev/uis0’ on the Linux system. As mentioned before, when the CIL core completes its operations, an interrupt signal is sent from the controller in the PL part. With a UIO driver, the processor can handle this interrupt signal by reading the device file. A Python application transmits the input data to the CIL core (PL) and receives the result from the CIL core. Additionally, the noise parameters are transmitted from the application. When the application is operating, it communicates with the CIL core by assigning the addresses of registers through the AXI interface. A simple example of the Python application using a register address is given below:

```
from lib.uio import Uio

class Pl:
    MSHOT_REGS = 0x00
    RW_REGS = 0x04
    # ...
    W1_REGS = 0x2C
    RESET_REGS = 0x44

    def __init__(self):
        self.uio = Uio('uis0', length=0x1000)
        self.regs = self.uio.regs()

pl = Pl()

# Write 1 to RESET register.
pl.regs.write_word(Pl.RESET_REGS, 1)
# Read from W1 register.
out_W1 = pl.regs.read_word(Pl.W1_REGS)
```

Through this design flow, the PS architecture of the SoC-based CIL hardware is described as shown in Fig. 9 (b).

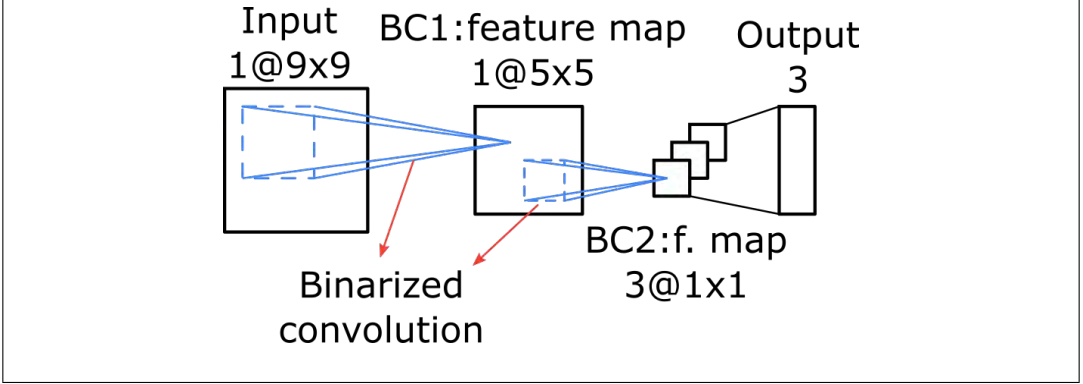


Figure 10: The 2-layer BCNN model used to test the proposed training hardware. It contains two binarized convolutional layers. The filter size of BC1 and BC2 is 5x5 pixels.

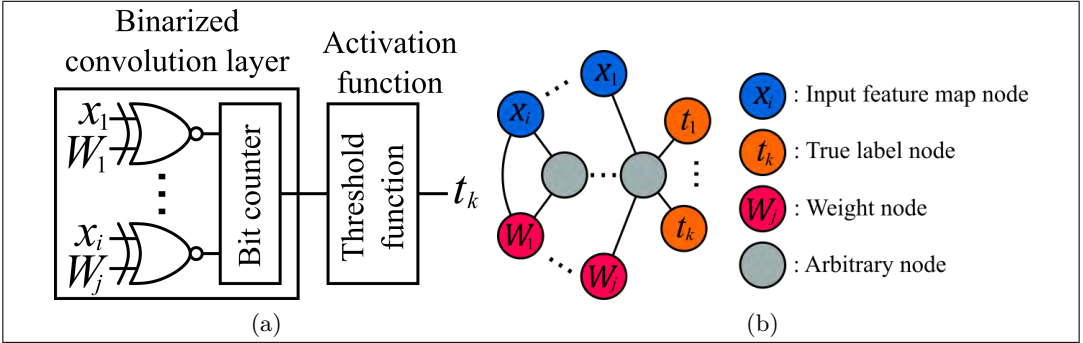


Figure 11: (a) The computation block for a binarized convolutional layer and the activation function of the BCNN model. (b) The configuration of the Hamiltonian for the training hardware. The nodes are categorized into input nodes, label nodes, weight nodes, and arbitrary nodes. On the basis of this Hamiltonian, the design of the CIL core is determined.

4 Evaluation

4.1 Case study: BCNN training hardware

For performance evaluation, we implemented training hardware for a 2-layer BCNN model [8] as a SoC-based prototype. Fig. 10 shows the 2-layer BCNN model that was used for the evaluation. It contains 2 binarized convolutional layers (BC1 and BC2) without biases, and its activation function is a threshold function. A training

dataset was organized by selecting ‘0’, ‘1’, and ‘5’ from the MNIST dataset [10], reducing the size of the images from 28x28 pixels to 9x9 pixels, and binarizing them. The details of the trained model and the dataset are described in [8].

The proposed training hardware for the 2-layer BCNN model is based on CIL; thus, the BCNN model must be converted into a Hamiltonian. The 2-layer model consists of binarized convolutional layers, so its inference computations can be implemented in logic circuits [21]. Fig. 11 (a) shows the computation block for a binarized convolutional layer and the activation function, which contains XNOR gates, a bit counter, and a threshold function. The components of this computation block can be converted into the corresponding Hamiltonian using ground-state spin logic. Thus, the Hamiltonian of the training hardware is obtained by combining the Hamiltonians of the components using the method shown in Fig. 8 (b). The Hamiltonian structure of the training hardware is shown in Fig. 11 (b); it consists of 2645 nodes, which are categorized into input nodes (x), label nodes (t), weight nodes (W), and arbitrary nodes. For the prototype training hardware based on CIL, the CIL core in the PL part was designed in accordance with this Hamiltonian.

When the training hardware operates, the input and label nodes are fixed as the training data and the true labels, respectively. Then, the weight and arbitrary nodes are driven to fluctuate by noise signals, and eventually, they reach stability when the energy converges to the minimum. The Hamiltonian of the training hardware is designed to take its minimum value when all nodes are in valid states, so the states of the weight nodes will correspond to the valid weights for the training data. However, the weights obtained in this way will be valid for only the one training data sample used to set the input and label nodes, not the entire dataset. The training hardware performs this process for each training sample to determine the final weights for the entire dataset. Let us use w to denote the individual weights from each training sample and N to denote the number of training samples in the dataset. The final weights for the entire dataset, W , are calculated as follows:

$$W = \frac{\sum_n^N w_n}{N}. \quad (7)$$

4.2 Test environment

The prototype hardware was implemented on a Xilinx Zynq-7000 ZC706 SoC FPGA board, and the embedded processor was an ARM Cortex-A9. Fig. 12 shows the test environment of the proposed SoC-based training hardware on the ZC706 FPGA board and the Python application on the embedded processor. The PL design was written in SystemVerilog and synthesized using Xilinx Vivado 2018.3. The power dissipation of the PL part is 0.732 W with a clock frequency of 12.5 MHz, and that

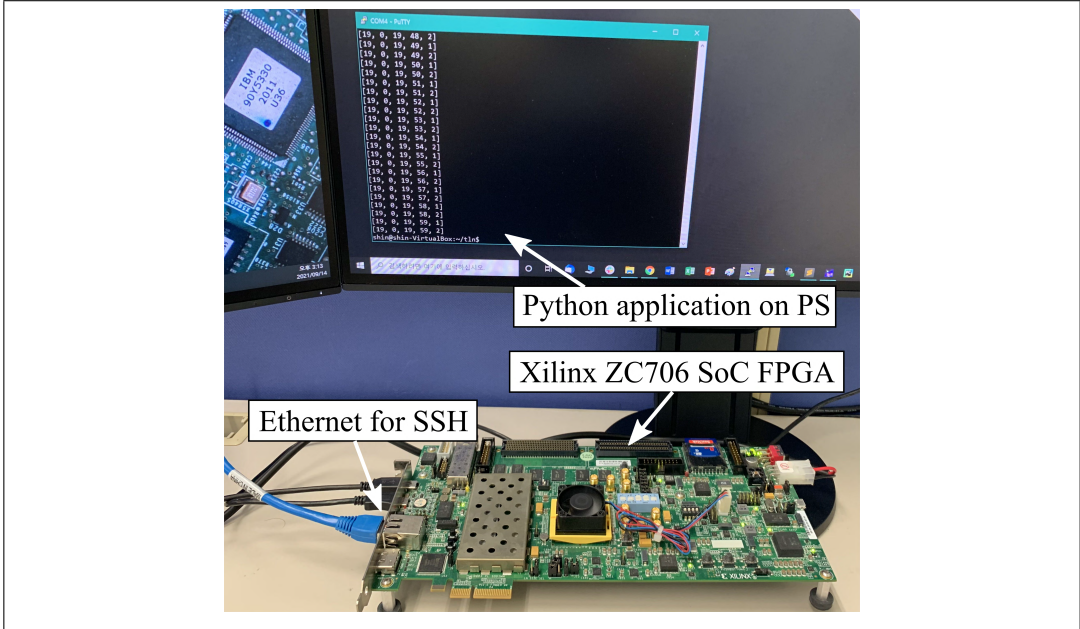


Figure 12: Test environment of the SoC-based CIL hardware.

Work	This work	FPGA-based hardware [8]
Clock frequency	12.5 MHz	
Utilization of LUTs	176501 / 218600 (80.74%)	170317 / 203800 (83.57%)
Utilization of FFs	43170 / 437200 (9.87%)	42951 / 407600 (10.54%)
Power dissipation of PS	1.566 W	-
Power dissipation of PL	0.732 W	0.913 W

Table 2: FPGA resource and power dissipation comparisons between the proposed SoC-based training hardware and the FPGA-based hardware.

of the PS is 1.566 W with a clock frequency of 667 MHz. The FPGA resource utilization and the power dissipation of the SoC-based hardware are summarized in Table 2, and they are compared with those of the FPGA-based hardware. Although the model trained here is only a 2-layer BCNN and the training dataset is quite small, the implemented Hamiltonian consists of 2,645 nodes, and both the SoC-based and FPGA-based training hardware implementations utilize almost all of the resources of the FPGA board. To implement training hardware for larger neural network models, such as LeNet-5, it would be necessary to convert from the neural network

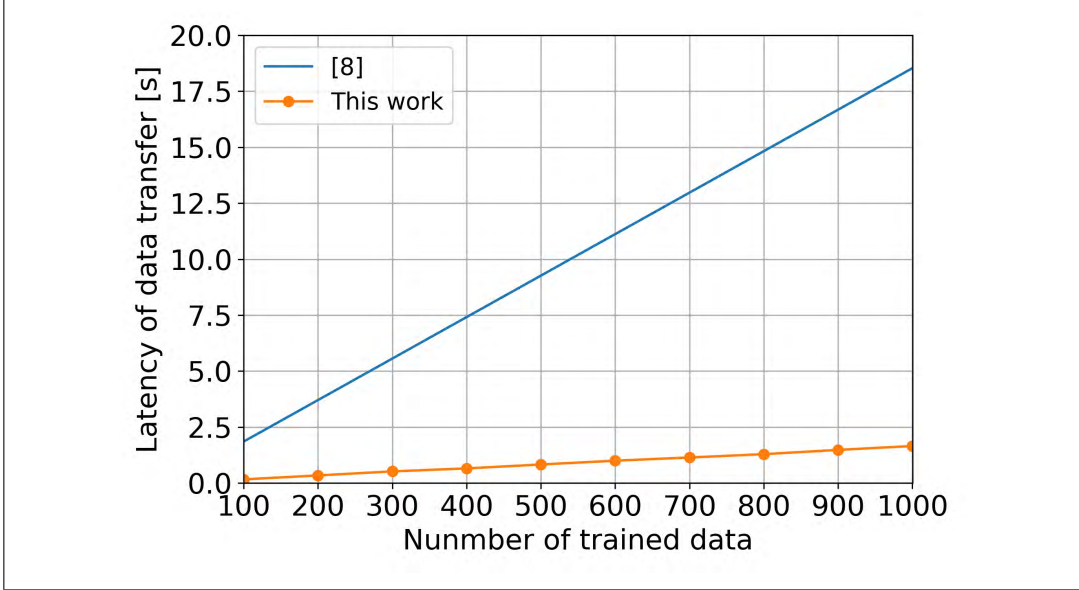


Figure 13: Comparisons of the data transfer latency between SoC-based and FPGA-based CIL training hardware implementations for a 2-layer BCNN model. The latencies of the FPGA-based hardware are estimates except for the case of training on 100 data samples. The data transfer latency of the SoC-based hardware is approximately 14.4x lower than that of the FPGA-based hardware.

model to a Hamiltonian that contains fewer nodes. The total power dissipation (PS + PL) of the SoC-based training hardware is higher than that of the FPGA-based hardware; however, the FPGA-based hardware also requires a processor, such as a PC, to execute the application. Therefore, in practice, the actual power dissipation of the FPGA-based hardware is higher than that of the SoC-based hardware.

The OS on the processor is Ubuntu 16.04, and the UIO driver and the application were written in Python 3.5. The SoC-based hardware is compared with the hardware from our previous work, which was implemented on a Digilent Genesys2 FPGA powered by a Xilinx Kintex-7. The clock frequency of this FPGA-based training hardware is 12.5 MHz, and it receives the training data via a UART interface.

4.3 Performance comparisons

The PL design of the SoC-based training hardware is the same as that of the FPGA-based training hardware; thus, we mainly compare the latency of data transfer. The training hardware treats a process spanning $2T + RS$ clock cycles as one shot in

regard to the noise signals [8]. Therefore, the number of clock cycles for which the hardware operates is given as follows:

$$N_{cycle} = N_{shot} * (2T + RS), \quad (8)$$

where N_{cycle} is the number of cycles and N_{shot} is the number of shots. The FPGA-based hardware was used to train the model on 100 data samples in 7883 shots, and the combination of noise parameters was $[RW, RS, T, MVAL, \alpha] = [17, 0, 53, 18, 2]$. In this case, the hardware operation time calculated from the number of clock cycles for which the hardware operates is 0.06685 s. However, the entire training time including data transfer is 1.92 s, indicating that the latency of data transfer is 1.85 s. In contrast, under the same conditions, the data transfer latency of the SoC-based hardware is 0.128 s, which is approximately 14.4x lower than that of the FPGA-based hardware. For training on one sample of training data, the training hardware receives a 9x9-pixel input image and a 3-bit true label and returns a 100-bit set of trained weights. The FPGA-based hardware communicates using a UART interface; therefore, data communication is performed 23 times. On the other hand, the proposed SoC-based hardware uses an AXI interface, which can handle 32 bits of data at once, so data communication is performed only 6 times. Thus, the SoC-based CIL hardware reduces the number of data communications and is more suitable for applications that need larger amounts of data.

The proposed training hardware based on CIL achieves a maximum cognition accuracy of 87.97% when using 100 training samples. The maximum accuracy of conventional training using backpropagation is 90.89%. Thus, the accuracy of the CIL-based training hardware is slightly reduced compared to that of conventional training. However, conventional training requires training on 10,800 samples of data 10 times to reach the maximum accuracy (max epochs = 10). Accordingly, the training time for conventional training is 2.68 s, which is 13.7x the training time of the proposed hardware.

Fig. 13 shows comparisons of the data latency between the SoC-based training hardware and the FPGA-based training hardware. The latency of the FPGA-based hardware is estimated except for the case of training on 100 data samples. When the FPGA-based hardware is used to train the model on 1000 samples, the data transfer latency is 18.5 s, which already represents a considerable bottleneck although the size of this training set is small compared to general datasets for training neural networks. In contrast, the measured latency of the SoC-based hardware in the case of training on 1000 data samples is 1.65 s, indicating that the SoC-based hardware suppresses the latency increase. Therefore, it is obvious that the SoC-based hardware is more suitable for data-consuming applications, such as training neural networks.

5 Conclusion

In this paper, we have introduced the prototyping of SoC-based CIL hardware for data-consuming applications, such as the training of BCNNs. FPGA-based CIL hardware receives its input data and noise parameters via a UART interface, which results in slow data transfer. Because the presented SoC-based hardware communicates with the embedded processor on the Zynq board, this enables suppression of the increase in the data transfer latency with an increasing amount of data to be communicated. The PL part of the SoC-based hardware was designed using a custom IP core, so it is easy to integrate into a SoC system. Additionally, the proposed hardware is a prototype, and the device driver was developed using a UIO interface, making it easy to modify. For performance comparisons, the SoC-based training hardware was implemented on a Xilinx ZC706 SoC board, and its data transfer latency was compared with that of an FPGA-based training hardware implementation. The SoC-based training hardware achieved approximately 14.4x faster data transfer than the FPGA-based hardware. Currently, the SoC-based hardware prototype has been implemented for training BCNNs. In the future, we will apply SoC-based CIL hardware not only for training BCNNs but also for other applications in which CIL is used.

Acknowledgments

This work was supported in part by JST PRESTO Grant Number JPMJPR18M5 and the WISE Program for AI Electronics, Tohoku University.

References

- [1] Kerem Yunus Camsari, Rafatul Faria, Brian M. Sutton, and Supriyo Datta. Stochastic p -bits for invertible logic. *Phys. Rev. X*, 7:031014, Jul 2017.
- [2] Geoffrey E Hinton, Terrence J Sejnowski, and David H Ackley. Boltzmann machines: Constraint satisfaction networks that learn. Technical Report CMU-CS-84-119, Dept. Comput. Sci. Pittsburgh, Carnegie-Mellon Univ., Pittsburgh, PA, USA, 1984.
- [3] R. Faria, K. Y. Camsari, and S. Datta. Low-barrier nanomagnets as p -bits for spin logic. *IEEE Magnetics Letters*, 8:1–5, 2017.
- [4] William A. Borders, Ahmed Z. Pervaiz, Shunsuke Fukami, Kerem Y. Camsari, Hideo Ohno, and Supriyo Datta. Integer factorization using stochastic magnetic tunnel junctions. *Nature*, 573(7774):390–393, 2019.
- [5] S. C. Smithson, N. Onizawa, B. H. Meyer, W. J. Gross, and T. Hanyu. Efficient CMOS invertible logic using stochastic computing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(6):2263–2274, June 2019.

- [6] B. R. Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 149–156, New York, NY, USA, 1967. ACM.
- [7] N. Onizawa, D. Shin, and T. Hanyu. Fast hardware-based learning algorithm for binarized perceptrons using CMOS invertible logic. *Journal of Applied Logics*, 7(1):41–58, Jan 2020.
- [8] Duckgyu Shin, Naoya Onizawa, Warren J. Gross, and Takahiro Hanyu. Training hardware for binarized convolutional neural network based on cmos invertible logic. *IEEE Access*, 8:188004–188014, 2020.
- [9] Duckgyu Shin, Naoya Onizawa, and Takahiro Hanyu. Fpga implementation of binarized perceptron learning hardware using cmos invertible logic. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 115–116, 2019.
- [10] Y. Lecun and C. Cortes. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>.
- [11] J. D. Biamonte. Nonperturbative k -body to two-body commuting conversion hamiltonians and embedding problem instances into ising spins. *Phys. Rev. A*, 77:052331, May 2008.
- [12] J. D. Whitfield, M. Faccin, and J. D. Biamonte. Ground-state spin logic. *EPL (Europhysics Letters)*, 99(5):57004, sep 2012.
- [13] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross. VLSI implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2688–2699, Oct 2017.
- [14] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel. Computation on stochastic bit streams digital image processing case studies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(3):449–462, March 2014.
- [15] B. D. Brown and H. C. Card. Stochastic neural computation. i. computational elements. *IEEE Transactions on Computers*, 50(9):891–905, Sep. 2001.
- [16] Umakanta Nanda and Sushant Kumar Pattnaik. Universal asynchronous receiver and transmitter (uart). In *2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS)*, volume 01, pages 1–5, 2016.
- [17] N. Onizawa, K. Nishino, S. Smithson, B. Meyer, W. Gross, H. Yamagata, H. Fujita, and T. Hanyu. A design framework for invertible logic. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 312–316, 2019.
- [18] Sebastiano Vigna. Further scramblings of marsaglia’s xorshift generators. *Journal of Computational and Applied Mathematics*, 315:175 – 181, 2017.
- [19] Grant Likely and Josh Boyer. A symphony of flavours: Using the device tree to describe embedded hardware. In *Proceedings of the Linux Symposium*, volume 2, pages 27–37, 2008.
- [20] Hans J Koch and H Linutronix Gmb. Userspace i/o drivers in a realtime context. In *The 13th Realtime Linux Workshop*, 2011.

- [21] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 525–542, Cham, 2016. Springer International Publishing.

EFFICIENT PAM-4 SYMBOL ESTIMATION USING SOFT CLUSTERING

YOSUKE IIJIMA

National Institute of Technology (KOSEN), Oyama College, JAPAN
yiiijima@oyama-ct.ac.jp

YASUSHI YUMINAKA

Graduate School of Science and Technology, Gunma university, JAPAN
yuminaka@gunma-u.ac.jp

Abstract

Multi-valued data transmission using four-level pulse amplitude modulation (PAM-4) is promising for alleviating the bandwidth limitation of interconnects. To evaluate the received signal quality during data transmission when using adaptive coefficient settings for a PAM-4 equalizer, we propose an eye-opening monitor method based on statistics of PAM-4 symbol transition. The proposed statistical eye-opening monitor is based on soft clustering and enables efficient PAM-4 data transmission under closed-eye conditions. Simulation results demonstrate the feasibility of symbol classification using the proposed statistical evaluation method.

1 Introduction

The rapid evolution of 5G mobile communications and cloud computing demands an exponential growth in data traffic, requiring ever-faster electrical interconnections between chips and underlying systems. However, owing to the increase in data rates, signal distortion due to intersymbol interference (ISI) and noise substantially limit the input/output (I/O) bandwidth. An electrical link with four-level pulse amplitude modulation (PAM-4) is often adopted to achieve high spectral efficiency and support the ever-increasing demands for data bandwidth. By conforming to next-generation high-speed data transmission standards such as IEEE 802.3bs, PAM-4 is a promising approach to support data transmission at rates up to 400 Gb/s (i.e., 8-lane transmission at 50 Gb/s) [1].

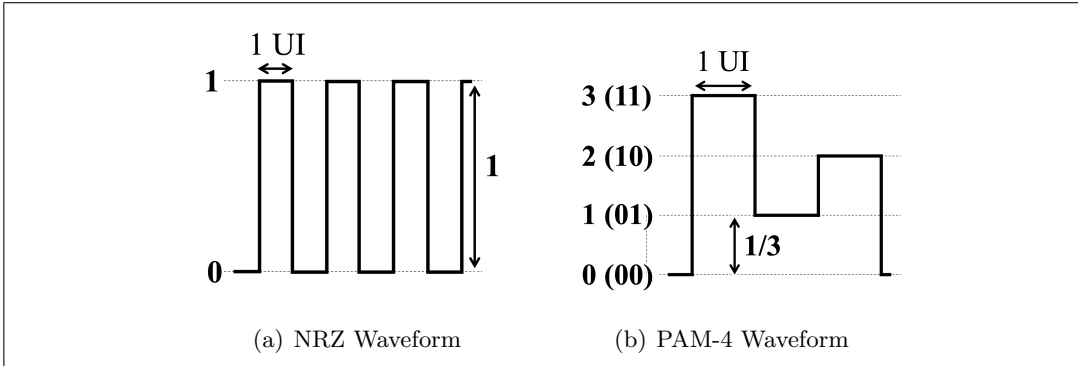


Figure 1: Transmitter waveforms of NRZ and PAM-4 signaling (UI, Unit Interval).

As shown in Fig. 1, PAM-4 can represent two consecutive binary non-return-to-zero (NRZ) bits using one symbol. Hence, PAM-4 can achieve a half symbol rate compared with NRZ for a given transmission rate. Although PAM-4 signaling can transmit data at the same baud rate using a twice-slower symbol rate, four-level signaling is more sensitive to the ISI and noise amplitudes than the NRZ signal.

To suppress ISI and improve the signal-to-noise ratio (SNR) at the receiver, various waveform shaping techniques are used in high-speed data transmission systems. A feed-forward equalizer (FFE) is a typical signal processing technique that can mitigate ISI at the transmitter [2]-[4]. To mitigate ISI at the receiver, analog-to-digital converter (ADC)-based PAM-4 receivers and equalization have recently been applied to perform flexible control using signal processing in the digital domain, as illustrated in Fig. 2 [5]-[7]. As this solution can be implemented using digital circuitry, the circuit parameters can be adaptively adjusted according to the ISI for the transmission line characteristics.

Accordingly, machine learning has been applied to statistical eye-opening monitors (EOM) for symbol detection from deteriorated transmitted PAM-4 signals to adjust the adaptive FFE coefficients [8]-[12]. Furthermore, using statistical methods may establish a new concept for symbol determination.

We propose a flexible symbol detection method based on the estimated symbol distributions. The proposed technique is introduced to demonstrate the feasibility of a new countermeasure for symbol deterioration. In conventional symbol detection based on threshold discrimination, it is difficult to correctly identify symbols under severe ISI, which causes the closed-eye condition. In [11], we estimated each symbol distribution for PAM-4 at a receiver using soft clustering. By obtaining the symbol distribution, a new symbol discrimination concept can be established. To detect a symbol correctly, we introduce PAM-4 symbol detection based on the symbol

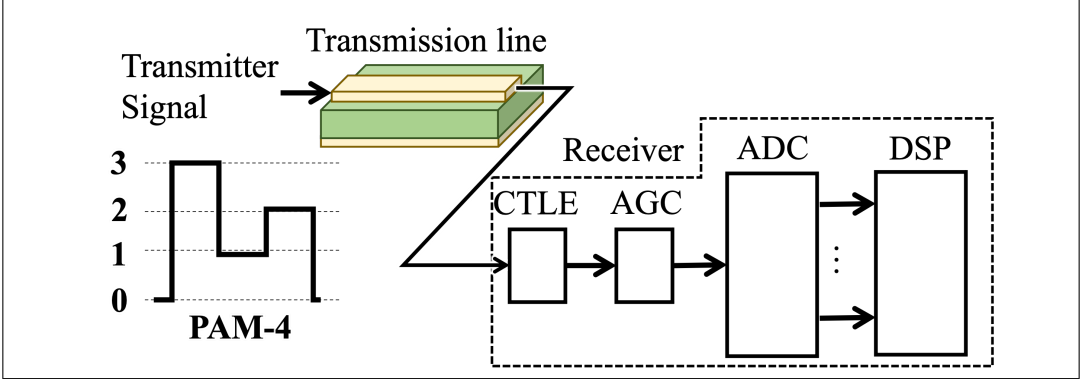


Figure 2: Overview of ADC-based PAM-4 receiver (CTLE, continuous-time linear equalizer; AGC, automatic gain controller; DSP, digital signal processor).

distribution estimation under closed-eye conditions [13]. Each symbol distribution of PAM-4 is first classified using a Gaussian mixture model (GMM), and then the detection of PAM-4 symbols is performed using soft clustering.

The remainder of this paper is organized as follows. Section 2 outlines the effects of waveform distortion on PAM-4 signals. In Section 3, we detail the proposed statistical EOM based on statistical characteristics of PAM-4 symbol transition. Section 4 presents the evaluation results of simulations to demonstrate the feasibility of the proposed EOM and the implementation analysis. Finally, Section 5 concludes the paper.

2 ISI in PAM-4 and statistical evaluation

2.1 ISI effect on PAM-4 signaling

As PAM-4 incorporates four signal levels, the amplitude per level is $1/3$ of a binary signal. Therefore, the effect of ISI on PAM-4 is higher than that on NRZ signaling with an increasing symbol rate. Moreover, transmitter linearity is important for PAM-4 signaling because nonlinear characteristics affect the equality of symbol distance at the receiver.

Figure 3 shows simulation results of eye diagrams for both NRZ and PAM-4 signaling at 1.0 Gb/s. An eye diagram allows to quantitatively evaluate the signal integrity using the eye height and width. The eye height and width correspond to the vertical and horizontal opening of the eye diagram, respectively. In addition, the eye aperture ratio can be evaluated using the eye height and width.

In simulations, we used the measured impulse response of a 2-m microstrip line

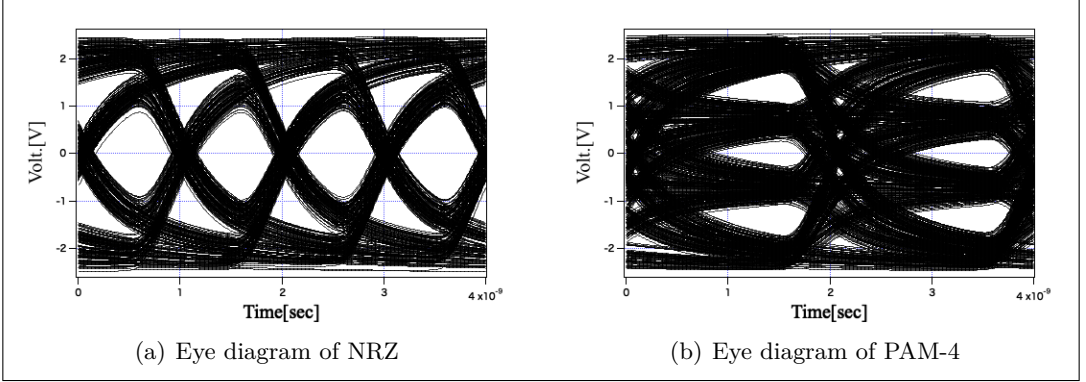


Figure 3: Simulated eye diagram of NRZ and PAM-4 signaling at 1.0 Gb/s on 2-m MSL.

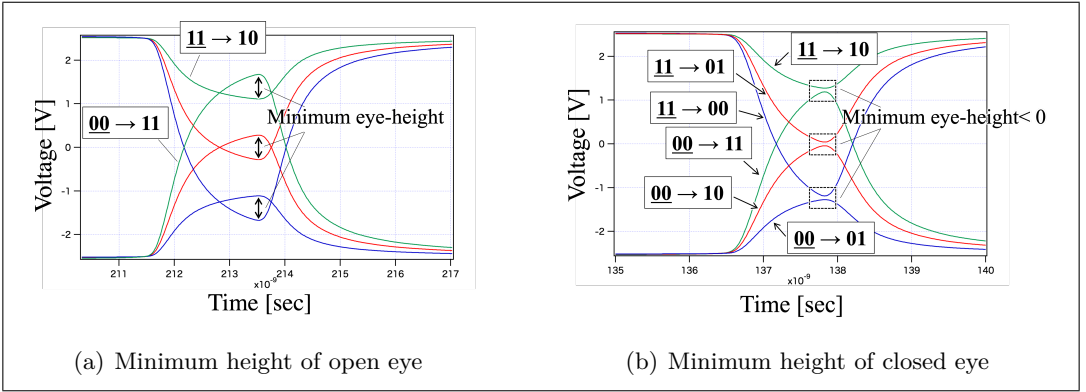


Figure 4: Minimum eye height in open and closed eye.

(MSL). As PAM-4 can transmit 2 bits per symbol, 1.0 Gb/s PAM-4 corresponds to a symbol rate of 0.5 GS/s (giga-symbol/sec). Therefore, PAM-4 can decrease the Nyquist frequency to half of that required for NRZ signaling. Despite the decreasing Nyquist frequency, the eye height of PAM-4 signaling is deteriorated to approximately 1/3 (−9.5 dB) compared with that of NRZ signaling. As the closed eye corresponds to zero eye height, the minimum eye height, which is the worst case for an open eye, is important for symbol estimation. For a closed eye, the received eye diagram closes theoretically because the minimum eye height becomes zero, impeding symbol classification using three threshold values (th_1 , th_2 , and th_3).

In PAM-4 signaling, the minimum eye height is affected by the symbol transition patterns. The minimum eye height of NRZ signaling is determined by two transitions (i.e., $0-1$ and $1-0$), whereas that of PAM-4 signaling is more complex. As shown

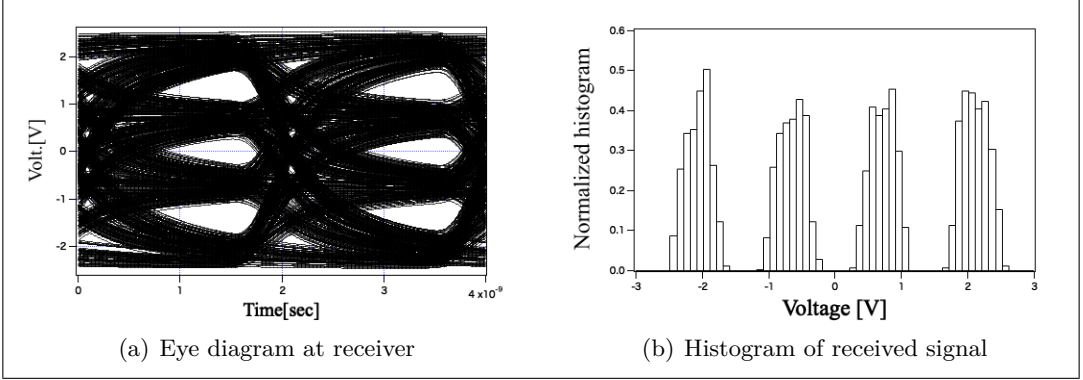


Figure 5: Simulation results of PAM-4 at 1.0 Gb/s on 2-m MSL.

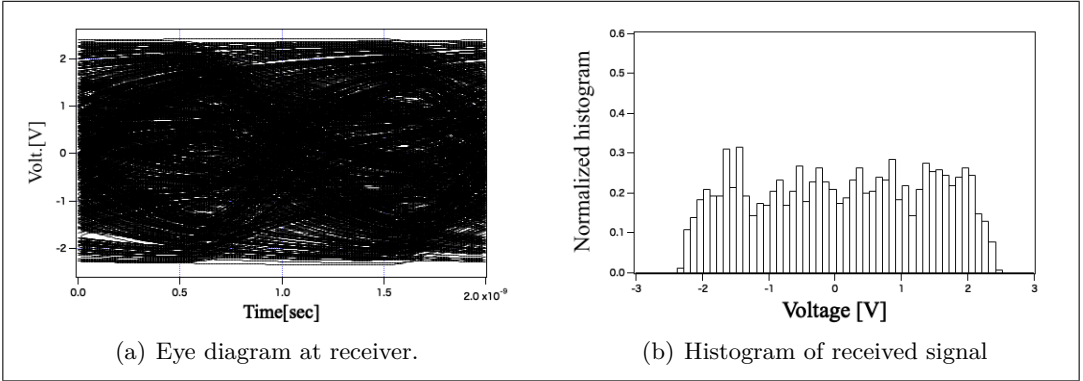


Figure 6: Simulation result of PAM-4 at 2.0 Gb/s on 2-m MSL.

in Fig. 4, the minimum PAM-4 eyes depend on six symbol transition patterns (i.e., **00–01**, **00–10**, **00–11**, **11–00**, **11–10**, and **11–01**). Therefore, the minimum eye height between symbols **10** and **11** is determined by transition patterns **00–11** and **11–10**. These patterns are the most affected in eye height between symbols **10** and **11**. On the other hand, in the minimum eye height between symbols **10** and **01**, symbol transition patterns **00–10** and **11–01** are the most affected.

The minimum eye height is important for accurate symbol classification. To classify a symbol correctly by threshold discrimination, a suitable minimum eye height must be guaranteed, and the eye aperture ratio should be improved. Hence, waveform shaping is necessary for closed-eye conditions to improve the minimum eye height to compensate for the channel loss, which increases with the data rate. However, when the eye is completely closed, quantitatively ISI evaluation is difficult at the receiver by using an eye-opening ratio, hindering waveform shaping.

2.2 Statistical evaluation using GMM

To evaluate ISI in closed-eye conditions, a statistical evaluation method using symbol distribution of the received signals has been proposed [13]. Figure 5 shows the eye diagram and histogram of received symbols for 1.0 Gb/s PAM-4. Using the histogram of the received symbol, ISI can be evaluated from the distribution of each symbol. The normalized histogram with a bin width of 0.1 V is shown according to the amplitude of the received signal. ISI can be characterized by the standard deviations and mean values of each symbol. Under mild ISI (Fig. 5(a)), the histogram has four separated distributions, and the symbol distributions are adequately separated, as shown in Fig. 5(b). Hence, when the eye is open at the receiver, each symbol (i.e., **00**, **01**, **10**, and **11**) can be classified correctly by using adequate thresholds. In addition, ISI can be statistically evaluated by measuring the mean and standard deviation from the histogram. In contrast, under severe ISI, it is difficult to distinguish each symbol due to the closing eye (Fig. 6(a)). As shown in Fig. 6(b), there is no space between contiguous distributions for 2.0 Gb/s PAM-4.

To evaluate the symbol distribution in the closing eye, soft clustering is introduced. Although it is difficult to judge the symbol correctly by threshold judgment in closed eye condition, it is possible to obtain the received symbol distribution by using soft clustering. Therefore, using the Gaussian mixture model, ISI can be evaluated quantitatively by estimating the symbol distribution using a GMM in the closing eye [11]. Assuming that the distribution of each symbol can be approximated as a Gaussian distribution, we can estimate four Gaussian distribution from the histogram of the received symbol. In this case, the distribution of the received PAM-4 signals is expressed as follows:

$$\begin{aligned}
 p(x) &= \sum_{k=0}^3 \pi_k N(x|\mu_k, \sigma_k) \\
 &= \pi_0 N(x|\mu_0, \sigma_0) + \pi_1 N(x|\mu_1, \sigma_1) \\
 &\quad + \pi_2 N(x|\mu_2, \sigma_2) + \pi_3 N(x|\mu_3, \sigma_3), \text{Equation1}
 \end{aligned} \tag{1}$$

where $N(x|\mu_k, \sigma_k)$ is a Gaussian distribution with mean μ_k and standard deviation σ_k , and π_k is the mixing coefficient corresponding to the weight for each Gaussian distribution. The sum across π_k ($\sum_{k=0}^3 \pi_k 1$) is 1. For example, $N(x|\mu_0, \sigma_0)$ indicates the distribution of symbol **00** with mean μ_0 and standard deviation σ_0 . As shown Eq. (1), the symbol distribution of PAM-4 can be expressed by the summation of four Gaussian distributions. Each symbol distribution of PAM-4 is estimated from the received signals by fitting Eq. (1) to the histogram of the received PAM-4 symbols.

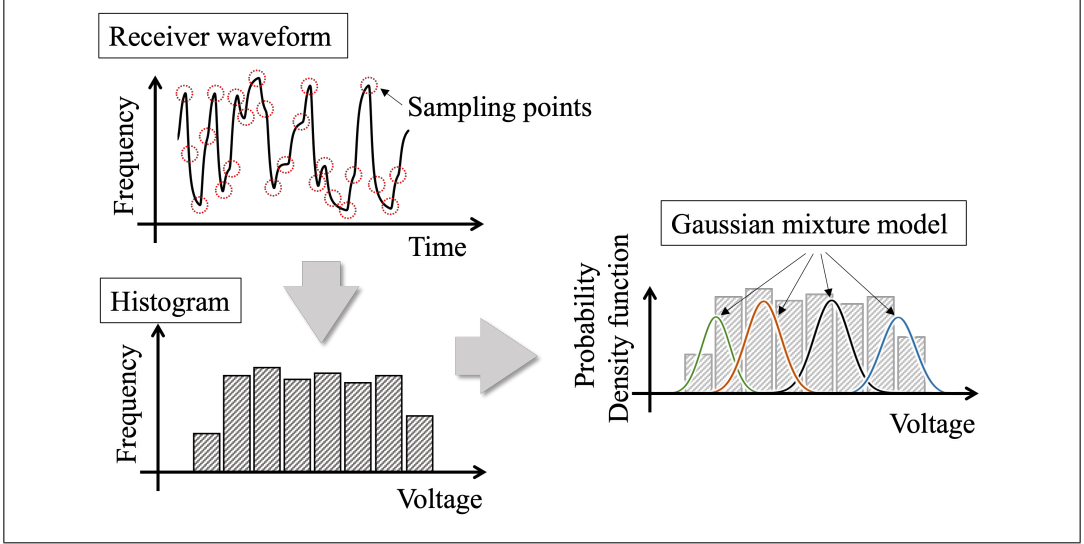


Figure 7: Eye-opening monitor using GMM evaluation.

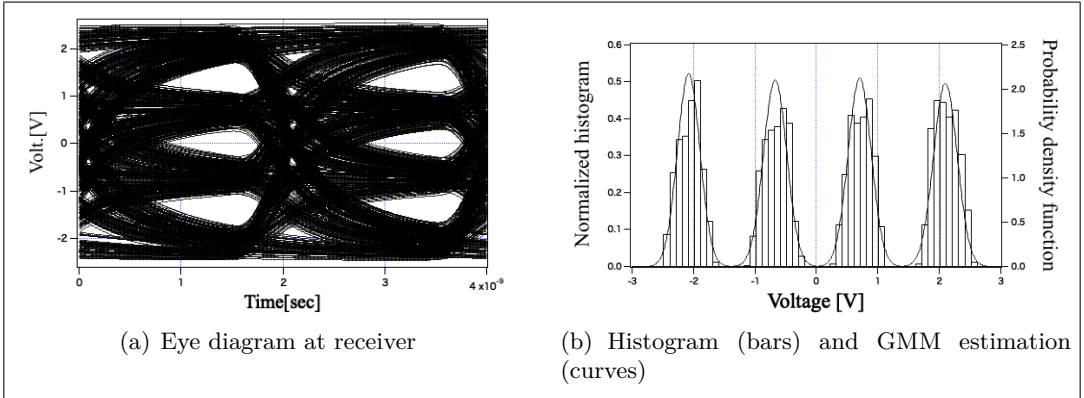


Figure 8: GMM estimation result of PAM-4 at 1.0 Gb/s on 2-m MSL.

As shown in Fig. 7, by fitting each symbol distribution from the histogram of sampling data at the receiver, ISI can be characterized statistically according to the standard deviation and mean of the distributions even when the eye is closed. In this study, we obtained the GMM parameters by applying the expectation-maximization (EM) algorithm [14].

Figure 8 shows the eye diagram and histogram of received symbols for 1.0 Gb/s PAM-4. In Fig. 8(b), the curves show the GMM fitting, with the left and right axes showing the normalized histogram and probability density function (PDF) for a bin

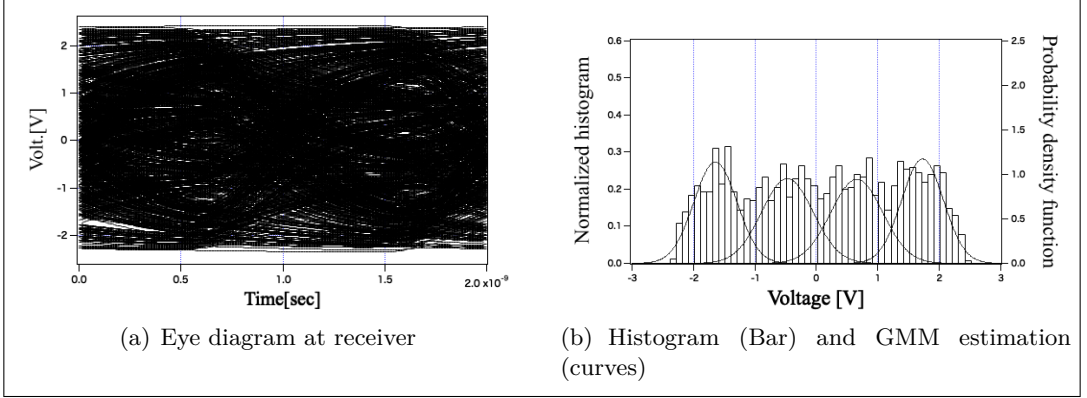


Figure 9: GMM estimation result of PAM-4 at 2.0 Gb/s PAM-4 on 2-m MSL.

width of 0.1 V, respectively. The GMM provides the distribution of each symbol. Figure 9 shows the eye diagram and histogram of received symbols for 2.0 Gb/s PAM-4. Although it is difficult to separate each symbol and evaluate each distribution, ISI can be estimated by fitting the symbol distributions using the histogram. By obtaining the distributions, ISI can be evaluated statistically according to standard deviation σ_k and mean μ_k of each distribution even when the eye is closed. The σ_k value indicates the ISI effect on each symbol. Moreover, the linearity of the transmitter can be evaluated using the μ_k value. Values σ_k and μ_k in Fig. 9(b) are $\{\sigma_0, \sigma_1, \sigma_2, \sigma_3\} = \{0.350, 0.418, 0.422, 0.340\}$ and $\{\mu_0, \mu_1, \mu_2, \mu_3\} = \{-1.644, -0.473, 0.656, 1.726\}$.

3 Symbol classification using statistical evaluation for PAM-4 signaling

3.1 Symbol classification using statistical evaluation with closed eye

To achieve symbol classification with closed eye, symbol detection using the symbol transition pattern based on GMM fitting is proposed. Although a symbol cannot be accurately classified by thresholding in closed eye owing to severe ISI, it can be classified under certain conditions by considering the PAM-4 characteristics.

Figure 9 shows that if symbol distributions overlap, classification errors may occur because multiple candidates coexist in the overlapping region. On the other hand, symbols can be correctly classified in nonoverlapping regions. For limited overlap, only adjacent symbols are mixed, and they can be detected as two candidates

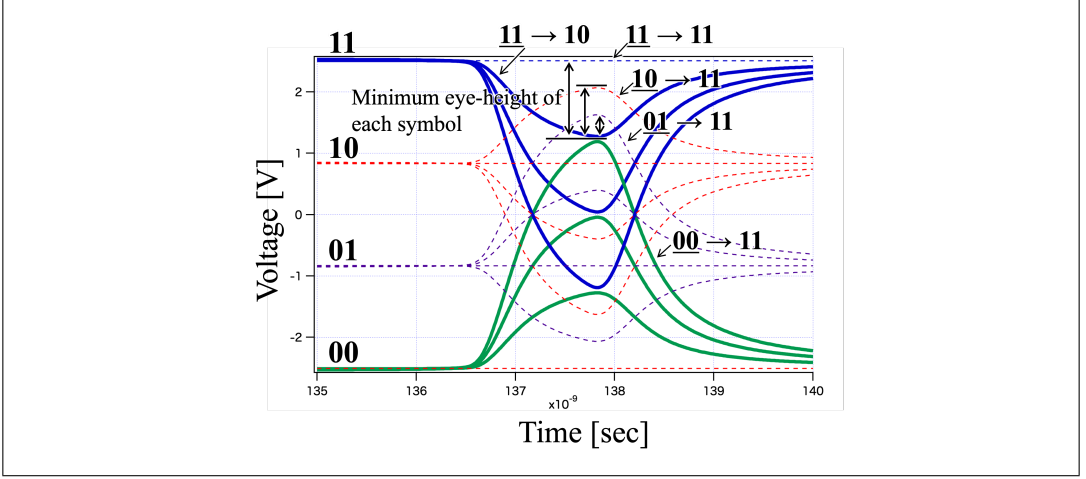


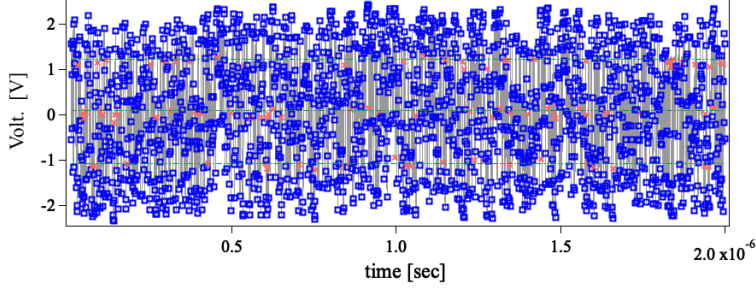
Figure 10: PAM-4 symbol transition at minimum eye height below 0 (solid lines, indistinguishable symbols; dashed line, classifiable symbols).

according to the PAM-4 characteristics.

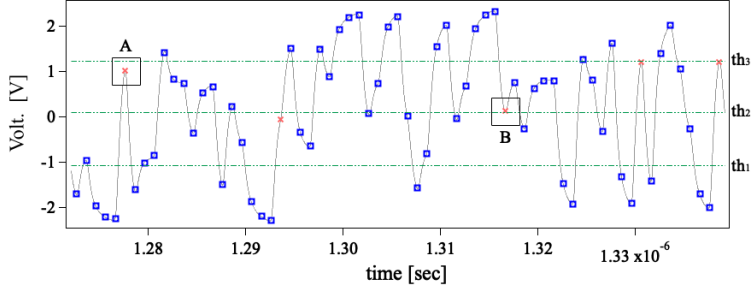
As shown in Fig. 10, although the PAM-4 eye aperture ratio is zero owing to ISI, symbol transitions **01-11**, **10-11**, and **11-11** can be separated from other transitions in this case. Therefore, a received symbol can be determined to be **11** with an adequate threshold. On the other hand, it is difficult to separate **00-11** from **11-10**. Similarly, symbol identification is possible for other symbol transitions when the eye aperture ratio is zero and the previous symbol is other than **00** or **11**, even for a minimum eye height below 0 with a certain influence of ISI.

Figure 11 shows the simulation results of symbol detection for 2.0 Gb/s PAM-4, which corresponds to the eye diagram shown in Fig. 9(a). In the simulation, the PRBS13Q symbol pattern was transmitted, and a micro-strip line with 16.0 dB attenuation at 1.0 GHz was used. In addition, the simulation was used to evaluate the ISI effect on the transmission line characteristics without additive noise. Using the impulse response of the transmission line, received waveform is simulated by data analysis software Igor Pro. In this simulation, no specific target and application were set, and the thresholds were determined to intermediate values between adjacent symbols.

These thresholds were calculated considering the junction points of the GMM curves shown in Fig. 9(b), obtaining thresholds th_1 , th_2 , and th_3 of -1.08 , 0.09 , and 1.23 V, respectively. As shown in Fig. 11, although most symbols can be detected correctly, some errors occur. Figure 11 shows that the symbol patterns of these errors are transitions from symbol **00** or **11**, whose patterns are strongly affected by ISI.



(a) Symbol detection with received waveform



(b) Magnified view

Figure 11: Simulation results of symbol detection for 2.0 Gb/s PAM-4 (blue square, correct classification; red cross, incorrect classification).

As shown in Fig. 11(b), received symbol A (transmitted symbol **11**) is incorrectly identified as **10** by thresholding owing to symbol transition **00-11** because it is impossible to classify between symbol patterns **00-11** and **11-10**. Similarly, received symbol B (transmitted symbol **01**) is incorrectly identified as **10** by thresholding owing to symbol transition **11-01** because it is hard to classify between symbol patterns **11-01** and **00-10**. In these examples, there are two candidates near the thresholds.

Nevertheless, the symbols can be correctly identified when the previous symbol is known, that is, **00-11** and **11-10** can be classified for previous symbol **00** or **11**. Specifically, when only adjacent distributions overlap in the receiver symbol distribution, the symbol can be determined if the previous symbol is known to be positive or negative, as illustrated in Fig. 10. Therefore, it is possible to identify the correct symbol from two candidates by using the information of the previous symbol. In

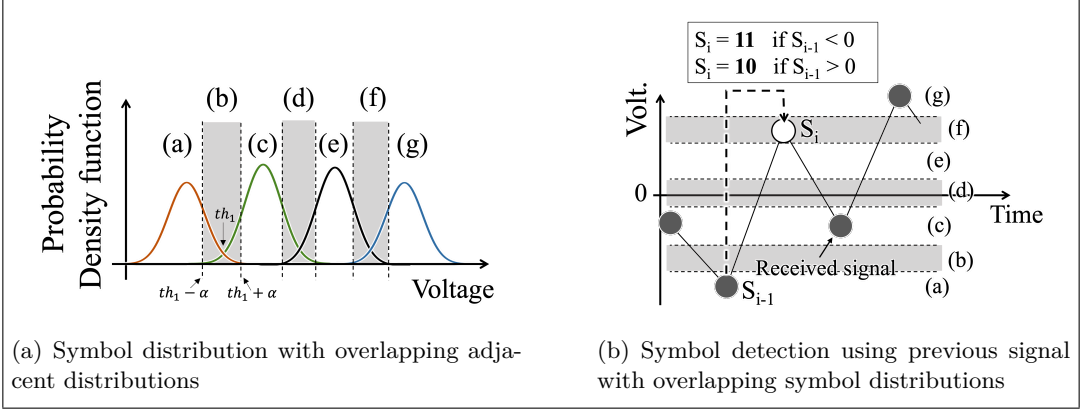


Figure 12: Diagram of symbol detection.

contrast, when distributions overlap beyond adjacent symbols, three or more symbol candidates exist. Consequently, symbol identification is impossible in principle. For example, if the symbol distribution of **00** overlaps with the distributions of **01** and **10**, three symbol candidates can appear.

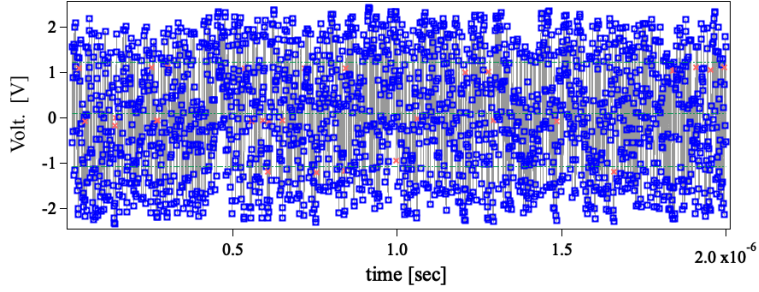
3.2 Symbol detection

The symbol distribution of PAM-4 can be divided into seven regions, a-g, as shown in Fig. 12(a). If signals within regions a, c, e, and g are received, the symbols are classified as **00**, **01**, **10**, and **11**, respectively. On the other hand, if signals within regions b, d, and f are received, the symbols should be identified according to the previous signal. As shown in Fig. 12(b), if received signal S_i is in region f, the symbol is decided depending on previous signal S_{i-1} . Specifically, S_i is symbol **11** if $S_{i-1} < 0$, and S_i is symbol **10** if $S_{i-1} > 0$.

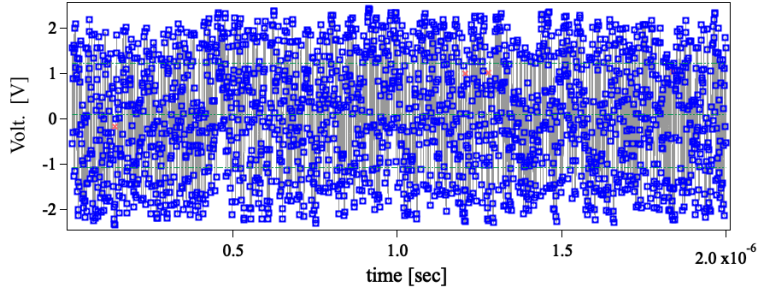
4 Simulation results and discussion

Figure 13 shows the simulation results of symbol detection with and without the proposed method for 2.0 Gb/s PAM-4. In [13], regions a-g were defined by $\mu_k \pm \sigma_k$ according to the values from the GMM. In Fig. 12, each region a-g was decided by th_k at the junction points of the GMM distributions. We introduce adjustment parameter α for each region to be set to $th_k \pm \alpha$ as shown in Fig. 12(a).

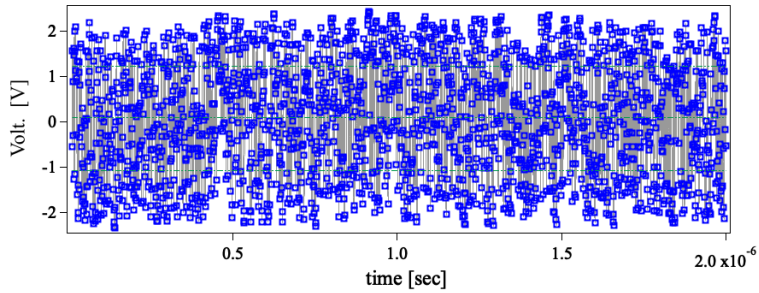
Figure 13 shows the simulation results for $\alpha = 0.1, 0.2$, and 0.3 . Values th_1 , th_2 , and th_3 were $-1.08, 0.09$, and 1.23 V, respectively. Compared with the results shown in Fig. 11, the proposed method can correctly detect ambiguous symbols



(a) Symbol detection and received waveform for $\alpha = 0.1$



(b) Symbol detection and received waveform for $\alpha = 0.2$



(c) Symbol detection and received waveform for $\alpha = 0.3$

Figure 13: Simulation results of proposed symbol detection for 2.0 Gb/s PAM-4 (blue square, correct classification; red cross, incorrect classification).

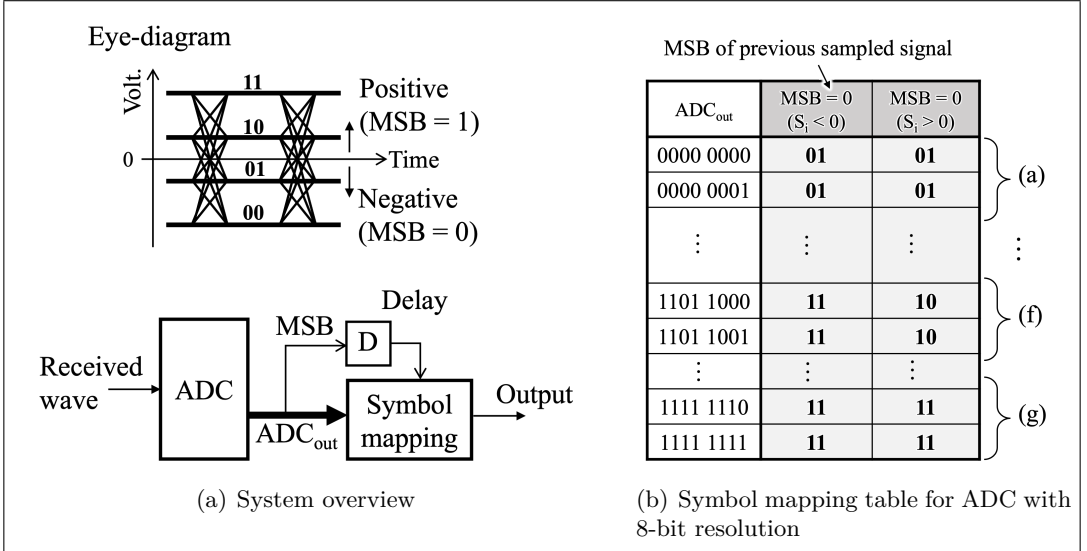


Figure 14: Overview of proposed symbol detection system (MSB, most significant bit).

with two candidates. Symbol errors occur in conventional thresholding, as shown Fig. 11, but they are substantially mitigated using the proposed symbol detection method without requiring waveform shaping. As shown in Fig. 13(c), symbol error is suppressed for $\alpha = 0.3$. The coarse range of each region is set using th_k , and the fine range can be adjusted by tuning α .

Figure 14 shows an overview of the proposed symbol detection system. The received signal is converted into a digital signal by an ADC, and symbol conversion is performed using digital processing according to the most significant bit (MSB) of the previous signal. As shown in Fig. 14(a), the MSB determines whether the previous symbol is the lower or upper two by setting the intermediate value of the input range of the ADC to the center value of the received waveform. Next, the received symbols are determined using a mapping table (Fig. 14(b)) based on regions a-g of the symbol distributions. In regions b, d, and f, the output symbol corresponds to the MSB of the previous symbol, indicating whether previous signal S_{i-1} is $S_{i-1} < 0$ or $S_{i-1} > 0$. In regions a, c, e, and g, the output symbol is decided regardless of the previous signal.

5 Conclusion

We proposed a novel concept for PAM-4 symbol estimation based on soft clustering of the symbol distribution. Using a GMM the threshold levels can be suitably set for classifying each PAM-4 symbol under unknown transmission characteristics. Moreover, we introduce PAM-4 symbol detection using previous symbol information. Owing to the optimization of the threshold levels, simulation results show classification of every symbol even when the eye is closed.

The proposed method can improve symbol detection without requiring waveform shaping techniques, such as pre-emphasis. Thus, the hardware implementation burden can be mitigated, and energy efficiency and bit error performance can be further improved by introducing waveform shaping.

To implement the proposed method, various aspects should be considered. For instance, to obtain an accurate GMM, the resolution of the ADC should be properly selected. In future work, we will conduct benchmark tests comparing various symbol detection methods and similar studies[15, 16]. In addition, experimental evaluation will be performed to determine the method performance regarding factors such as high-speed transmission rates.

References

- [1] IEEE P802.3bs 200 Gb/s and 400 Gb/s Ethernet Task Force, <http://www.ieee802.org/3/bs/>
- [2] G. Steffan, E. Depaoli, E. Monaco, N. Sabatino, W. Audoglio, A.A. Rossi, S. Erba, M. Bassi, and A. Mazzanti, "A 64Gb/s PAM-4 Transmitter with 4-Tap FFE and 2.26pJ/b Energy Efficiency in 28nm CMOS FDSOI," *2017 IEEE International Solid-State Circuits Conference*, pp.116–117, 2017.
- [3] T.O. Dickson, H.A. Ainspan, and M. Meghelli, "A 1.8pJ/b 56Gb/s PAM-4 Transmitter with Fractionally Spaced FFE in 14nm CMOS," *2017 IEEE International Solid-State Circuits Conference*, pp.118–119, 2017.
- [4] Altera: Whitepaper for Using Pre-Emphasis and Equalization with Stratix GX Device, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp_pre-emphasis.pdf
- [5] Y. Iijima and Y. Yuminaka, "Double-Rate Tomlinson-Harashima Precoding for Multi-Valued Data Transmission," *IEICE Transactions on Information & Systems*, vol.100-D, no.8, pp.1611–1617, 2017.
- [6] Y. Iijima and Y. Yuminaka, "Waveform Shaping Transmitter Combining Digital and Analog Circuits for Multi-Valued Signaling," *Proc. IEEE 49th International Symposium on Multiple-Valued Logic*, pp.19–24, 2019.

- [7] D. Cui et al., “3.2 A 320mW 32Gb/s 8b ADC-based PAM-4 analog front-end with programmable gain control and analog peaking in 28nm CMOS,” *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 58-59, doi: 10.1109/ISSCC.2016.7417905.
- [8] N. Sato, T. Chigira, K. Toyoda, Y. Iijima, and Y. Yuminaka, “Multi-Valued Signal Generation and Measurement for PAM-4 Serial-Link Test,” *Proc. IEEE 48th International Symposium on Multiple-Valued Logic*, pp.210–214, 2018.
- [9] B. Analui, A. Rylyakov, S. Rylov, M. Meghelli, and A. Hajimiri, “A 10-Gb/s Two-Dimensional Eye-Opening Monitor in 0.13- μ m Standard CMOS,” *IEEE Journal of Solid-State Circuits*, vol.40, no. 12, pp.2689–2699, 2005.
- [10] Y. Yuminaka, T. Kitamura, and Y. Iijima, “PAM-4 Eye Diagram Analysis and Its Monitoring Technique for Adaptive Pre-Emphasis for Multivalued Data Transmissions,” *Proc. IEEE 47th International Symposium on Multiple-Valued Logic*, pp. 13-18, 2017.
- [11] Y. Iijima, K. Taya, and Y. Yuminaka, “PAM-4 Eye-Opening Monitoring Techniques using Gaussian Mixture Model,” *IEEE 50th International Symposium on Multiple-Valued Logic*, pp.149–154, 2020.
- [12] Y. Iijima and Y. Yuminaka, “PAM-4 Eye-Opening Monitor Technique Using Gaussian Mixture Model for Adaptive Equalization,” *IEICE Transactions on Information & Systems*, vol.104-D, no.8, pp.1138–1145, 2021.
- [13] Y. Iijima and Y. Yuminaka, “Efficient PAM-4 Data Transmission with Closed Eye Using Symbol Distribution Estimation,” *IEEE 51st International Symposium on Multiple-Valued Logic*, pp.195–200, 2021.
- [14] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2009.
- [15] J. Kim, A. Balankutty, R. Dokania, A. Elshazly, H-S. Kim, S. Kundu, D. Shi, S. Weaver, K. Yu, and F. O’Mahony, “A 112 Gb/s PAM-4 56 Gb/s NRZ Reconfigurable Transmitter With Three-Tap FFE in 10-nm FinFET,” in *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp.29-42, Jan. 2019, doi: 10.1109/JSSC.2018.2874040.
- [16] J. Im et al., “A 112Gb/s PAM-4 Long-Reach Wireline Transceiver Using a 36-Way Time-Interleaved SAR-ADC and Inverter-Based RX Analog Front-End in 7nm FinFET,” *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 116-118, doi:10.1109/ISSCC19947.2020.9063081.

DATA-CLASSIFICATION-BASED DETERMINATION FOR OPHTHALMOLOGICAL EXAMINATION CATEGORIES USING MACHINE LEARNING

SHOJI MORITA^{†, ††}, TEIJIRO ISOKAWA^{††}, NAOTAKE KAMIURA^{††}

[†]Glory Ltd

^{††}Graduate School of Engineering, University of Hyogo

`morita.shoji@mail.glory.co.jp`,

`{isokawa, kamiura}@eng.u-hyogo.ac.jp`

HITOSHI TABUCHI

Department of Ophthalmology, Saneikai Tsukazaki Hospital

*Department of Technology and Design Thinking for Medicine, Hiroshima
University*

`h.tabuchi@tsukazaki-eye.net`

Abstract

In this paper, a method for determining examination categories is proposed for ophthalmology patients, using machine learning. It is assumed that the examinations are classified into four categories. The discrimination models constructed using machine learning are applied to determine which group each patient's medical questionnaire belongs to. The target to be classified is Japanese sentences handwritten by the patients in their questionnaires. The proposed method mainly consists of morphological-analysis step, vector-preparation step, and machine-learning step. In the first step, either MeCab or Sudachi is employed to decompose character strings into parts of speech. In the second step, one of the following means are conducted to assign values to elements in the vectors corresponding to the questionnaires: One Hot Encoding, Bag of Words, and Word2Vec. In the final step, one of the following machine-learning schemes is conducted: Support Vector Machine, CatBoost, and Neural Networks. Experimental results show that a combination of Sudachi, One Hot Encoding and CatBoost is favorable to achieve the highest accuracy in determining the examination category.

1 Introduction

Recently, medical institutions in Japan have faced problems on the long waiting time imposed on outpatients for consultations and/or examinations, and the long sojourn time that outpatients must consume at them. Such problems seem to be one of reasons why outpatients hesitate to attend medical institutions. Besides, it is considered that shortening the waiting time and sojourn time is one of the key factors that strongly affect rating for medical institutions.

As a countermeasure for overcoming the problem associated with the waiting time and sojourn time, the approach introducing electronic medical records [1], [2] and the approach based on the queuing theory [3] are proposed to directly shorten them. Utilizing the waiting time effectively is considered to be another countermeasure. For example, examinations for which medical doctors are not required to be there are imposed on outpatients during their waiting time at many medical institutions. Medical doctors can generally give directions on examinations, which should be imposed on outpatients when they will next attend medical institutions, using their medical records, in the cases where they keep attending the institutions. On the other hand, new outpatients must consume the extra time until their examinations start, in addition to the waiting time imposed on outpatients that keep attending the institutions. This is because medical doctors must determine examinations that should be imposed on the new outpatients after they read medical questionnaires that the new outpatients filled out. The rate of the length of effectively available time compared to the total length of waiting time thus tends to decrease for the new outpatients. If it is possible to automatically determine examinations necessary for the new patients according to the information written by them in the medical questionnaires, the above extra time required for the doctors to read the questionnaires can be drastically reduced. The system determining examinations as mentioned above will make it possible to remarkably decrease the amount of stress that both medical doctors and outpatients feel.

In this paper, a method of determining examination categories is proposed for ophthalmology patients filling out their medical questionnaires. It focuses on sentences in the questionnaires handwritten by the patients to be examined, and prepares data, which are presented to discrimination models constructed by machine learning, from the sentences. It considers that examinations valid for representative eye diseases can be divided into four examination categories, and that the patients absolutely take examinations belonging to one of the four categories. The trained models classify the data corresponding to the questionnaires. The proposed method thus copes with the determination of examination categories as the classification of the questionnaires. To decompose the handwritten sentences into parts of speech, it

applies either MeCab [4] or Sudachi [5], which is known as tool for the morphological analysis. It next assigns values to the words, which are picked up by the above analysis tool and considered to be elements useful in characterizing the sentences handwritten by each patient, conducting one of the following means: One Hot Encoding, Bag of Words, and chiVe [6] considered to be Word2Vec in which Sudachi is utilized. To construct discrimination models, it is assumed that one of machine learning algorithms, support vector machine (SVM for short) [7], CatBoost [8], or neural network (NN for short) [9], is available in the proposed method. A model is trained with the data prepared in the above manner. When the examination category for some patient is determined, his/her data, which is prepared in the manner similar to the preparation of training data, is presented to the trained model. In this paper, the choices of morphological analysis tools, vector preparation manners, and machine learning algorithms are examined to acquire the powerful capability in determining examination categories.

The methods of determining examination categories using data prepared from medical questionnaires are also proposed in [10] and [11]. SVM learning and NN learning are applied in [10] and [11], respectively. From experimental results, it is finally revealed that a combination of Sudachi, One Hot Encoding, and CatBoost is useful in achieving the high accuracy in the case where a small-scale dataset is available.

2 Preliminaries

2.1 Classes of medical questionnaires

The proposed method determines types of examinations, referring to sentences handwritten in medical questionnaires. In Saneikai Tsukazaki Hospital, eye diseases diagnosed by ophthalmologists are divided into thirteen categories. There however exists cases where some type of examinations is required to two or more eye diseases belonging to separate categories. In this paper, all of the eye diseases are categorized into four groups as shown in Table 1, and it is assumed that an outpatient filling out his/her medical questionnaire takes examinations belonging to one of the groups in Table 1. In other words, it can be considered that the number of classes of medical questionnaires is four as shown in Table 1. Note that Table 1 shows the correspondence between the class and representative eye diseases belonging to it.

The proposed method employs keyboard input to generate computerized information on sentences handwritten in medical questionnaires. Revisions are then conducted according to the following policies.

Classes	Categories	Diseases
1	Mydriasis-related examinations	Cataract, Diabetes, Retinal disease, Uveitis, Pediatric ophthalmology
2	Glaucoma-related examinations	Glaucoma
3	Examinations for anterior ocular segments	Strabismus, Neuro-ophthalmologic disease, Trauma
4	Oculomotor-related examinations	Corneal and/or conjunctival disease, Lacrimal apparatus, Ametropia, The others

Table 1: Relationships between classes and diseases

1. Clear typographical errors appearing in handwritten sentences are revised.
2. Words associated with inconsistent dates are revised.
3. There are cases where two or more different Chinese characters (i.e., kanji) have the same meaning. For example, two kanji characters are available to express “eye.” In such cases, the usage of characters is unified with the identical character chosen from them.

2.2 Tools for morphological analysis

Sentences written in Japanese and Chinese generally have no word boundaries. If these sentences are analyzed, it is not easy to estimate the size of an output sequence. This is a quite large difference between Japanese sentences and sentences written in other languages in labeling sequences for the morphological analysis. Determining word boundaries is thus of importance in analyzing Japanese sentences.

In this paper, two tools known as MeCab and Sudachi are applied for the morphological analysis. MeCab employs conditional random fields (CRF for short), which are the model for recognition developed to label sequences. Using CRF makes it possible to conduct learning so that right sequence labeling can be discriminated from other candidates of sequence labeling. Sudachi is the tool disclosed in 2017. Since it is a newcomer compared with MeCab, it has the new dictionary available for the powerful different notation normalization. As an example of the different notation normalization, “syumilation,” which is often used by Japanese people wrongly, is converted to “simulation.” kanji characters are often used in the substitute for other kanji characters having the same pronunciation. For example, “fuzoku” meaning the word “be attached to” in English has two types of notations using kanji characters. The dictionary for Sudachi also copes well with such cases. Similar sounding/different character cases often appear in handwritten sentences to be treated in this

paper. The notation normalization made by Sudachi seems to work well for the proposed method.

2.3 Discrimination models

In this paper, the following three types of machine learning are discussed to construct discrimination models: SVM, CatBoost, and NN. There are big differences among them in terms of algorithm structures. SVM learning constructs discrimination models according to the margin maximization principle. It basically chooses the hyperplane running through the middle points between two-class data among planes that can perfectly divide the data into the two groups. This gives the powerful generalization capability to constructed discrimination models. When SVM learning is conducted on condition that the Lagrange multiplier is used, the learning process results in solving the two-dimensional optimization problem. The frequency of producing the local minimum for SVM learning therefore tends to be low compared with that for CatBoost learning and NN learning. In this paper, the nonlinear SVM based on radial basis function (RBF for short) kernel is applied.

Catboost learning consecutively generates decision trees as weak learners. Figure 1 illustrates examples of such decision trees. It is considered that Catboost learning is one of the gradient boosting algorithms, which can combine inference results of the weak learners. In other words, the gradient boosting algorithm generates a new decision tree, referring to a dataset and the result of decision tree that has been the most recently generated. Since an approximate gradient value is then acquired from the same dataset, it is possible that the difference between the actual probability distribution and the probability distribution on predicted gradient values occurs as the prediction shift. Catboost learning employs ordered boosting, which samples a new dataset each time decision trees are generated, to overcome the prediction-shift problem and to reduce overfitting.

In this paper, a simple fully connected NN as shown in Figure 2 is introduced. A numerical vector is presented from the “Input” located in the left-most part, while the right-most part “Output” produces a numerical vector. “FC” means a fully connected layer. The first FC consists of 128 neurons. The number of neurons in the second FC is 4. This number is equal to the number of classes shown in Table 1. Note that $u = 128$ and $u = 4$ denote numbers of neurons. Rectified linear unit (ReLU) activation function and Softmax activation function are denoted by “Relu” and “Softmax,” respectively. Besides, to reduce overfitting, the dropout scheme is employed in the proposed method. The dropout scheme randomly and temporarily deletes one of the neurons in the intermediate layer to cut the signal propagation during learning. “ $r = 0.9$ ” in Figure 2 means the dropout rate to be 0.9.

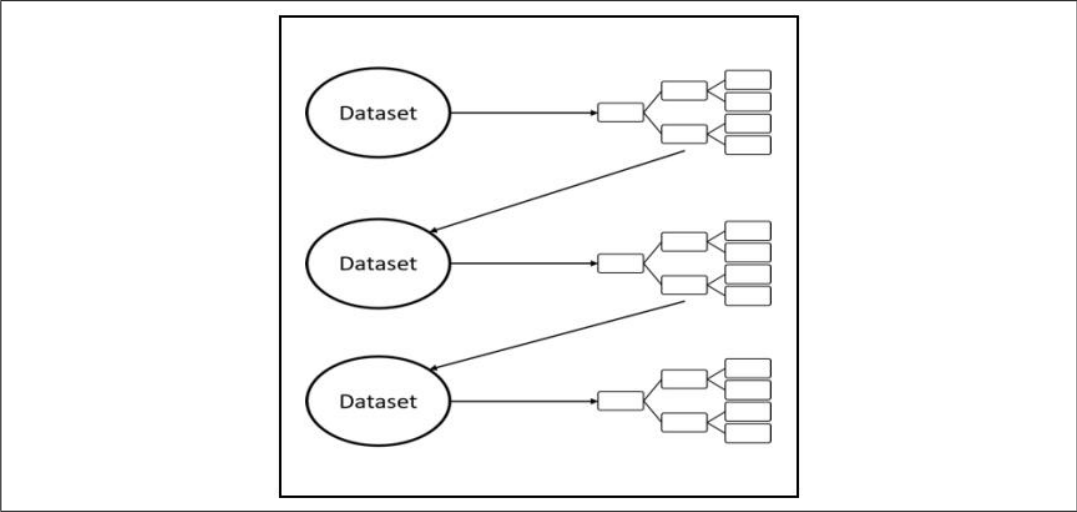


Figure 1: Decision trees generated by Catboost

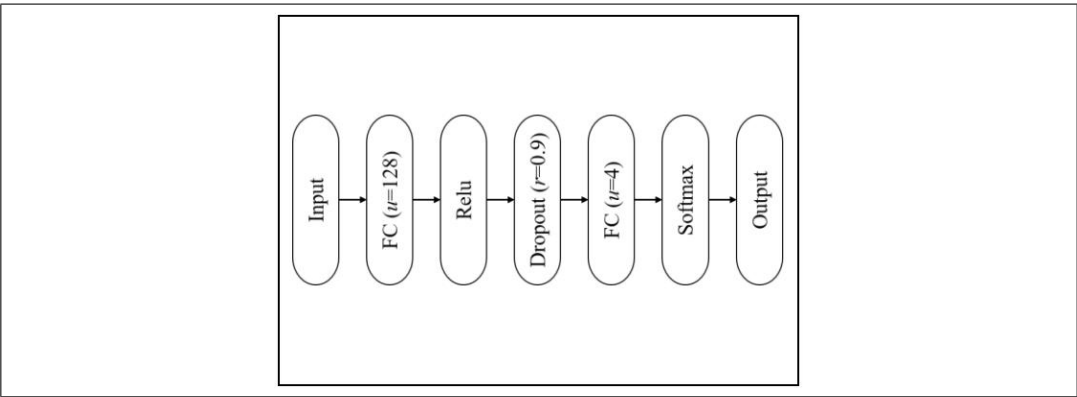


Figure 2: Structure of NN

3 Determination of examination categories using machine learning

There are the following three types of characters in Japanese: hiragana, katakana, and kanji. In addition, some conjugations are available. The character types and conjugations have generally made it more complicated to apply the morphological analysis in Japanese. For example, when some character string is easily decomposed into parts of speech, synonymous words (e.g., the verb “miru,” which is translated as “see,” written with hiragana characters solely and the verb “miru” written

	眼にボールが当たった (a) Original sentence	
眼, に, ボール, が, 当たっ, た (b) Result when level 1 list is employed	目, に, ボール, が, 当たる, た (c) Result when level 2 list is employed	メ, ニ, ボール, ガ, アタル, タ (d) Result when level 3 list is employed

Figure 3: Example of differences that depend on lists

with hiragana characters together with a kanji character) are often considered to be independent words. The similar failures in decomposing strings often occur for differences caused by their conjugations (e.g., “iku” translated as “go” and “itta” translated as “went”). The proposed method simultaneously generates the following three word lists: the list having results only acquired using the morphological analysis tools, the list having archetypes associated with conjugations of results acquired using the morphological analysis tools, and the list having the archetypes which are transformed from results of the morphological analysis and written using katakana characters. The first, second, and third lists are hereinafter referred to as the level 1, level 2, and level 3 lists, respectively. Let us briefly discuss the difference that occurs in the cases where these lists are employed, using “me ni bohru ga atatta” as an example. This sentence written in Japanese has kanji characters (i.e., “me” and “ata”), hiragana characters (i.e., “ni,” “ga,” and “tta”) and katakana characters (i.e., “bohru”), and it is translated as “the ball hit my eye.” When the level 1 list is employed, it is decomposed as follows: “me, ni, bohru, ga, atattu, ta” expressed with kanji, hiragana, and katakana characters. Employing the level 2 list results in “me, ni, bohru, ga, ataru, ta.” The decomposition result is also expressed with kanji, hiragana, and katakana characters. In the case of employing the level 3 list, we have “me, ni, bohru, ga, ataru, ta” expressed with katakana characters solely. Figure 3 illustrates the above.

To construct discrimination models for medical questionnaires using machine learning, it is necessary to prepare numerical vectors from sentences handwritten in them. The proposed method eventually employs one of the following means: One Hot Encoding, Bag of Words, and Word2Vec. When employing either One Hot Encoding or Bag of Words, the proposed method prepares a two-dimensional matrix with N rows and d columns, using the above lists, and considers element values in each of the rows to be one of the data presented to the discrimination models. A row then corresponds to a medical questionnaire. The number of patients for which examination categories have been known is therefore equal to N . On the other hand, the columns are for members of a word set specified by the above lists in addition

to the age of each of the N patients. The proposed method thus characterizes a medical questionnaire with a d -dimensional vector. Note that the age of the patient is given to the element corresponding to it without processing.

Let us explain the case where One Hot Encoding is employed for the matrix with N rows and d columns. When a word appears (or does not appear) in the sentences handwritten in some medical questionnaire, the value of 1 (or 0) is given to the element where the row and the column respectively corresponding to the questionnaire and the word cross. On the other hand, in the case of employing Bag of Words, the numbers of appearance are given as element values for words. For example, when some word is handwritten two times in some questionnaire, the value of 2 is given to the element where the row and the column respectively corresponding to the questionnaire and the word cross. Note that the value of 0 is given to any element for words not appearing in the questionnaire.

Word2vec transforms a word according to a distributed vector representation with its specific dimensionality. In this paper, chiVe, which is considered to be advanced Word2vec based on the skip-gram algorithm [12], is employed. The skip-gram algorithm solves a problem on predicting words appearing around a given word, and learns distributed representations for every word. If either One Hot Encoding or Bag of Words is employed, the number of elements in each of the data tends to be large, because it severely depends on the number of words involved in the list. On the other hand, employing Word2vec makes it possible to reduce the number of elements, compared with employing the above two means. Besides, since learning is completed with a large-scale sentence set in advance for Word2vec, Word2vec seems to adequately transform the words, which only appear in the questionnaires to be classified to determine examination categories.

The proposed method produces a hundred-dimensional vector for each of the words, employing chiVe. It then uses archetypes of the words. The archetypes are obtained by Sudachi that chiVe utilizes for learning. The amount of sentences handwritten in medical questionnaires clearly differ from each other. If a word is always converted to a hundred-dimensional vector, the dimension number of a vector corresponding to a questionnaire is equal to the number of words appearing in it multiplied by a hundred. This causes much of a difference in the amount of characteristics between vectors corresponding to questionnaires. To overcome this problem, the proposed method introduces four schemes. The first scheme estimates the mean values for each of element values in hundred-dimensional vectors corresponding to the words appearing in each questionnaire. This estimation prepares a hundred-dimensional vector for a questionnaire, and the vector is used in machine learning as data for the corresponding questionnaire. This data-preparation scheme is hereinafter referred to as Word2Vec-mean. The second (or third) scheme obtains

the maximum (or minimum) value for each of element values in hundred-dimensional vectors corresponding to the words appearing in each questionnaire, and prepares a vector with the maximum (or minimum) values of a hundred elements for a questionnaire. The second (or third) preparation scheme is hereinafter referred to as Word2Vec-max (or Word2Vec-min). The fourth scheme uses a pair of vectors prepared by Word2Vec-max and Word2Vec-min. In other words, it concatenates them to prepare a two-hundred-dimensional vector for a questionnaire. It is hereinafter referred to as Word2Vec-minmax.

The proposed method is evaluated, while changing a combination of morphological-analysis tools, word lists, and machine learning algorithms. Let us consider the case where Word2vec is not employed. The following choices are then available: MeCab or Sudachi for morphological analysis, level 1, level 2, or level 3 for word list, One Hot Encoding or Bag of Words for preparing a dataset (i.e., a two-dimensional matrix with N rows and d columns), and SVM, CatBoost, NN for the learning algorithm. The total number of combinations for the above choices is 36. The method proposed in [11] corresponds to the determination of examination categories using MeCab, level 3 list, One Hot Encoding and NN. On the other hand, in the case of employing Word2vec, Sudachi and chiVe are employed for morphological analysis and word-list generation, respectively. The following choices are then available: Word2Vec-mean, Word2Vec-max, Word2Vec-min, Word2Vec-minmax for preparing a dataset, and SVM, CatBoost, NN for the learning algorithm. The total number of combinations is therefore 12, when Word2vec is employed. Figure 4 depicts the above combinations used in the proposed method.

4 Experimental results

4.1 Machine-learning-based classification of medical questionnaires

As mentioned above, thirty-six approaches are available when Word2vec is not employed, whereas the number of available approaches is 12 when Word2vec is employed. In this section, forty-eight approaches are evaluated in terms of accuracies of determining examination categories. If SVM is chosen as a learning algorithm, the grid search is applied to determine parameters associated with the kernel function. For the NN structure shown in Figure 2, AMSgrad is applied as an optimizer algorithm. Besides, learning rate, batch size, and epoch number are set to 0.001, 128, and 250, respectively. The ten trials of evaluation are conducted, while randomly changing initial parameters of a discrimination model every trial.

The sentences handwritten in medical questionnaires were provided from Department of Ophthalmology in Saneikai Tsukazaki Hospital. They are for outpatients

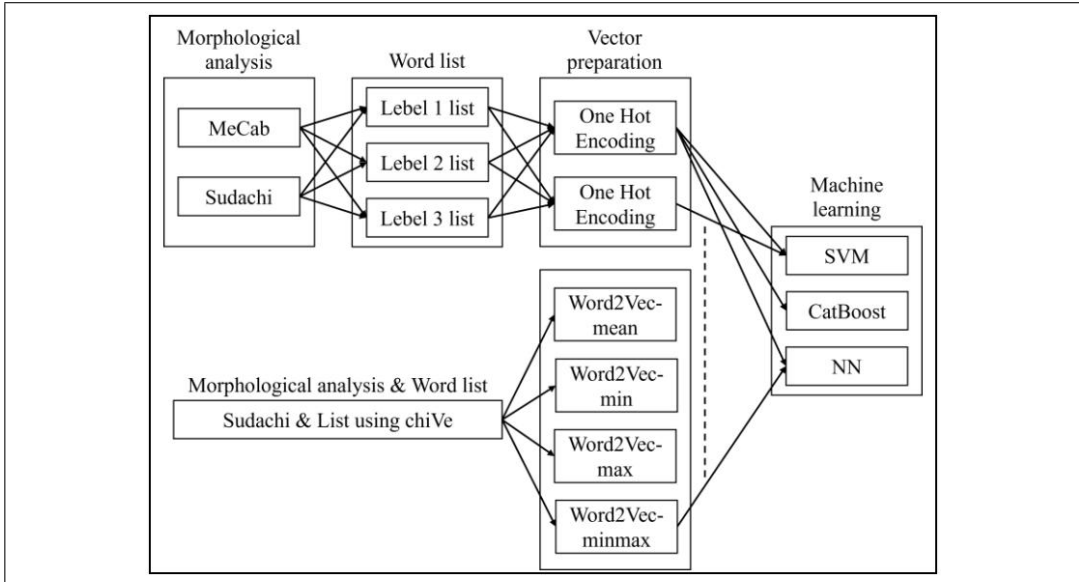


Figure 4: Combinations used in proposed method

that first visited Tsukazaki Hospital for the period from May through November 2010. The data were prepared from the handwritten sentences, and were divided into a training dataset and a test dataset. The breakdown of members in the datasets is tabulated in Table 2. Note that class numbers 1 through 4 correspond to those in Table 1, and that a member in the dataset corresponds to data prepared from sentences handwritten in a medical questionnaire. The classification results for members in the test dataset are used in evaluating discrimination models.

The evaluation results are tabulated in Table 3. Each of the entries in Table 3 is the percentage of the number of test data of which classes are correctly judged, compared to the total number of test data. The proposed method achieves the most favorable accuracy, 76%, under the combination of Sudachi for morphological analysis, level 2 list, Bag of Words for vector preparation, and SVM learning to construct discrimination models. Detailed classification results under this combination are tabulated in Table 4. The entry value appearing in the cell where the row of actual class i and the column of judged class j cross is equal to the percentage of the number of data judged as class j compared with the number of data actually belonging to class i .

Classes	1	2	3	4
Numbers of members in a training dataset	120	120	120	120
Numbers of members in a test dataset	25	25	25	25

Table 2: Breakdown of members in datasets

Morphological analysis		MeCab			Sudachi		
Machine learning		SVM	CatBoost	NN	SVM	CatBoost	NN
Word list	Vector preparation						
Level 1 list	One Hot Encoding	69.0	69.0	71.0	71.0	69.8	74.6
	Bag of Words	65.0	68.8	69.8	68.0	68.7	71.9
Level 2 list	One Hot Encoding	70.0	68.3	70.5	72.0	73.1	72.9
	Bag of Words	62.0	68.8	68.8	76.0	72.0	71.1
Level 3 list	One Hot Encoding	67.0	70.6	75.0	75.0	72.0	73.9
	Bag of Words	69.0	70.3	72.5	73.0	71.3	71.5
List using chiVe	Word2Vec-mean	—	—	—	52.0	62.4	52.8
	Word2Vec-min	—	—	—	59.0	69.8	53.1
	Word2Vec-max	—	—	—	52.0	62.0	52.8
	Word2Vec-minmax	—	—	—	60.0	68.6	57.4

Table 3: Results for test data shown in Table 2

4.2 Comparison with other determination methods

Let us briefly explain the determination of examination categories proposed in [10]. The method in [10] employs level 1 list, MeCab for morphological analysis and SVM learning to construct discrimination models. It defines prohibited words as the words not to be included in the word list. It also defines appearance rate as the number associated with appearances of words, and MD-designated words as words that ophthalmologists consider to be included in the word list. In addition to MD-designated words, the method in [10] adds the words with 5.5% or more as appearance rates to the word list. It basically prepares a two-dimensional matrix with N rows and d columns, according to Bag of Words. It then weights all entries

Judgement \ Actuality		Judged classes			
		1	2	3	4
Actual Classes	1	72.0	16.0	4.0	8.0
	2	16.0	76.0	4.0	4.0
	3	0.0	0.0	96.0	4.0
	4	24.0	8.0	8.0	60.0

Table 4: Detailed classification results when best accuracy is achieved

Judgement Actuality		Judged classes			
		1	2	3	4
Actual Classes	1	71.6	20.0	4.4	4.0
	2	26.0	64.8	7.6	1.6
	3	10.0	1.2	86.8	2.0
	4	16.0	3.6	22.4	58.0

Table 5: Detailed classification results obtained by method in [10]

		Judged class			
		1	2	3	4
Actual Classes	Class 1	66.8	26.4	6.8	0.0
	Class 2	15.6	75.6	4.0	4.8
	Class 3	2.4	1.6	92.0	4.0
	Class 4	12.4	6.4	9.6	71.6

Table 6: Detailed classification results obtained by method in [11]

on some columns if words corresponding to the columns are either MD-designated words or words fulfill the conditions specified by the appearance rates. The matrix has a column corresponding to ages of the outpatients. If the age of some outpatient is less than 11 years-old, the value of 0 is given to the element where the row corresponding to the outpatient and the column cross. If the age belongs to the range [11, 45], the value of 6 is given. If the age is over 46, the value of 12 is given. For SVM learning, RBF function is employed as the kernel function, and parameters are determined by the grid search. The detailed classification results obtained by the method in [10] are tabulated in Table 5. Recall that the method in [11] is based on the combination of MeCab, level 3 list, One Hot Encoding, and NN. The detailed classification results obtained by the method in [11] are tabulated in Table 6.

To estimate the judgements conducted by ophthalmologists, quizzes were introduced as follows in [10]. The seven ophthalmologists working at Saneikai Tsukazaki Hospital determined the examination categories for the outpatients handwriting sentences in medical questionnaires, which correspond to test data in Table 2, after reading the sentences. Note that the number of quizzes is 100. The results were averaged for the seven ophthalmologists. The averaged accuracies are tabulated in Table 7. The value averaged for four entries in Table 7 is equal to 55.7. In a department of ophthalmology, results of consultation and various examinations are employed in addition to medical questionnaires, finally to estimate disease names and to fix treatment plans. Though medical questionnaires are of importance, they are one of the ophthalmologist's decision tools. To determine examination categories

Averaged classification results [%]			
Class 1	Class 2	Class 3	Class 4
65.7	42.3	77.7	37.1

Table 7: Averaged classification results achieved by seven ophthalmologists in [10]

on the basis of only reading the handwritten sentences seems to be a very hard task for the ophthalmologists. This is why comparatively low results appear in Table 7.

Let us first compare entries in Table 4 with those in Table 5. Note that each of the entries on diagonal lines in Tables 4-6 equals the accuracy for each class. The proposed method using Sudachi, level 2 list, Bag of Words, and SVM learning achieves higher accuracy for every class than the method in [10]. In [10], MD-designated words, appearance rates, and weighting values on some columns are introduced according to the knowledge of ophthalmologists. While the proposed method drastically reduces the usage of the medical knowledge, it improves the averaged accuracy by 5% or more compared with the method in [10]. Note that the difference between class-1 and class-4 accuracies in Tables 4 and 5 is comparatively small. It seems that the strategies of defining prohibited words, weighting all entries on some columns in a two-dimensional matrix, and setting appearance rates work well for checking test data actually belonging to classes 1 and 4. On the other hand, such strategies would affect the classification for the test data to be classified as class 2. The method in [10] thus has difficulty of appropriately using the medical knowledge. It is specially hard to choose adequate MD-designated words.

The entries in Table 4 are next compared with those in Table 6. Though the class-4 accuracy achieved by the proposed method is lower than that achieved by the method in [11], the former achieves favorable accuracies for the other classes compared to the latter. Comparing the accuracies in Table 4 with those in Table 7 finally clarify that the former accuracies are higher than the latter accuracies for all classes. Note that the similar advantage also applies to other methods in [10] and [11] in terms of accuracies averaged for all classes. As mentioned in Section 1, medical doctors must determine examinations that outpatients first visiting their hospitals must take, after the doctors read medical questionnaires filled out by the new outpatients. Since the classification made by the proposed method is almost as high as that made by an average ophthalmologist, it is expected that the proposed method can reduce the time that the average ophthalmologist requires to read the questionnaire and to determine the examination category, if the proposed method is applied. On the other hand, the classification results tabulated in Table 4 are so low that the proposed method can never be applied to diagnose diseases.

4.3 Evaluation based on cross-validation

For Tables 3-6, the proposed method is evaluated on condition that both training data and test data are fixed as shown in Table 2. Besides, the amount of data leaves much to be desired. It is therefore possible that the discrimination models overfit the fixed dataset with the breakdown shown in Table 2. In this subsection, the proposed method is evaluated using the five-fold cross-validation. The total number of data is 580. Note that it equals the number of data in Table 2. The five combinations each of which consists of 464 training data and 116 test data were generated, and accuracies averaged for the five trials using them were acquired as results using the five-fold cross-validation. For the evaluations in this subsection, the parameters on learning are equal to those determined for the evaluations discussed in Subsects. 4.1 and 4.2. The averaged results are tabulated in Tables 8-11. Each of entries in Table 8 is the percentage of the number of test data of which classes are correctly judged, compared to the total number of test data. In other words, Table 8 corresponds to Table 3. The most favorable accuracy, 64.3%, was estimated under the combination of Sudachi for morphological analysis, level 3 list, One Hot Encoding for vector preparation, and CatBoost-based discrimination model. Detailed classification results under this combination is tabulated in Table 9. In addition, detailed classification results obtained by the methods in [10] and [11] are tabulated in Tables 10 and 11, respectively. In them, note that the entry appearing in the cell where the row of actual class i and the column of judged class j cross is equal to the percentage of the number of data judged as class j compared with the number of data actually belonging to class i .

Let us discuss the comparison of entries in Tables 9 and 10. The accuracies achieved by the proposed method are slightly lower than those achieved by the method in [10] for classes 1 and 3, while the proposed method is superior to the method in [10] on accuracies for classes 2 and 4. The proposed method therefore copes well with determining examination categories compared with the method in [10] in terms of the accuracy averaged for all classes by 5% or more.

The similar advantage applies when entries in Table 9 are compared with those in Table 11. In other words, though class-1 and class-3 accuracies achieved by the proposed method are lower than those achieved by the method in [11], employing the proposed method makes it possible to achieve high class-2 and class-4 accuracies compared with employing the method in [11]. As a result, the proposed method improves the accuracy averaged for all classes by about 3% compared with the method in [11].

The direct comparison in terms of accuracies between the seven ophthalmologists and the proposed method seems to be unfair, because the proposed method was

Morphological analysis		MeCab			Sudachi		
Machine learning		SVM	CatBoost	NN	SVM	CatBoost	NN
Word list	Vector preparation						
Level 1 list	One Hot Encoding	60.7	62.9	62.2	60.2	63.1	61.6
	Bag of Words	57.6	62.9	60.6	57.2	62.9	60.1
Level 2 list	One Hot Encoding	61.7	63.9	61.5	58.1	63.6	61.5
	Bag of Words	58.3	63.9	61.6	56.7	63.3	61.5
Level 3 list	One Hot Encoding	59.1	63.4	61.8	58.8	64.3	61.7
	Bag of Words	58.3	62.9	61.1	55.3	64.0	61.0
List using chiVe	Word2Vec-mean	—	—	—	50.9	57.9	54.5
	Word2Vec-min	—	—	—	47.6	58.1	51.1
	Word2Vec-max	—	—	—	48.6	57.4	53.0
	Word2Vec-minmax	—	—	—	52.8	58.5	54.8

Table 8: Results using five-fold cross-validation

Judgement Actuality		Judged classes			
		1	2	3	4
Actual Classes	1	69.8	13.7	9.8	6.8
	2	21.6	61.6	9.7	7.2
	3	8.7	10.2	64.1	17.0
	4	13.7	8.0	16.6	61.7

Table 9: Detailed classification results when best accuracy is achieved under five-fold cross-validation

evaluated using cross-validation and the ophthalmologists were tested using the fixed dataset. Though this unfairness is taken into account, let us dare to compare entries in Table 7 with those in Table 9. The difference between the ophthalmologists and the proposed method is large in terms of the class-3 accuracy. The accuracies shown in Table 9 are substantially higher than those in Table 7 for each of the other classes. The accuracy averaged for all classes estimated from Table 9 therefore exceeds that from Table 7.

5 Discussions

The accuracies achieved by the proposed method for classes 1-4 are 72%, 76%, 96%, and 60%, respectively, as shown in Table 4. Thus, the proposed method specially works well with the class-3 test data. Let us briefly discuss the characteristics of members in the class-3 test dataset. The average number of words appearing in questionnaires corresponding to test data to be classified as classes 1, 2, 3, and 4 are

Judgement Actuality		Judged classes			
		1	2	3	4
Actual Classes	1	71.2	9.5	12.8	6.4
	2	32.9	50.3	11.0	5.9
	3	15.5	5.8	67.2	11.4
	4	14.8	4.1	38.7	42.4

Table 10: Detailed classification results obtained by method in [10] under five-fold cross-validation

Judgement Actuality		Judged classes			
		1	2	3	4
Actual Classes	1	79.3	5.5	11.0	4.1
	2	31.7	53.8	10.3	4.1
	3	20.7	6.2	66.2	6.9
	4	21.4	12.4	20.7	45.5

Table 11: Detailed classification results obtained by method in [11] under five-fold cross-validation

15.52, 16.24, 11.16, and 16.68, respectively. In addition, the following three words can be picked up by exploring words included in all test data in terms of frequencies of their appearance: “ryokunaisho,” “hakunaisho,” and “nen.” Note that the first, second, and third words are translated as “glaucoma,” “cataract,” and “year,” respectively. The word “glaucoma” and/or the word “hakunaisho” appear seven, eleven, zero, and two times in questionnaires corresponding to classes 1, 2, 3, and 4 test data, respectively. The outpatients handwriting sentences in questionnaires to be classified as classes 1, 2, and 4 generally have past medical histories, and hence there are many cases where such patients tend to describe concrete disease names in the questionnaires. The word “nen” appears seven, two, zero, and two times in questionnaires corresponding to classes 1, 2, 3, and 4 test data, respectively. The outpatients often handwrites it to describe past points in time when their symptoms occurred (e.g., “san nen mae” translated as “three years ago”). The number of words expressing concrete matters such as disease names and symptoms is thus smaller in questionnaires to be classified as class 3 than that of words handwritten in questionnaires to be classified as any other class. The above seem to be the reason why the proposed method can achieve very high accuracy for class-3 test data.

The proposed method is evaluated in Subsect. 4.2, subject to using fixed datasets as shown in Table 2, while the five-fold cross-validation is introduced for the evaluation in Subsect. 4.3. Let us first compare the combination making it possible

to achieve the highest accuracy in Subsect. 4.2 with that in Subsect. 4.3. Recall that, on condition of using the fixed datasets in Subsect. 4.2, the proposed method achieves the highest accuracy, 76.0%, averaged for all classes under the following combination: Sudachi, level 2 list, Bag of Words, and SVM learning. The five-fold cross-validation results in 56.7% as the averaged accuracy when the proposed method is conducted under the same combination. The difference between the case of using the fixed datasets and that of introducing the cross-validation is about 20%. It seems that the degree of overfitting becomes heavier as this difference becomes larger. The method based on the combination of Sudachi, level 2 list, Bag of Words, and SVM learning would be useless in reducing the overfitting situation.

On the other hand, for the cross-validation-based evaluation, a combination of Sudachi, level 3 list, One Hot Encoding, and CatBoost learning seems to be the most promising. The accuracy averaged for all classes is then equal to 64.3%, and it is high compared with the accuracy calculated under any other combination. The proposed method using this promising combination, however, constructs a discrimination model with 72.0% as the accuracy averaged for all classes when members in the fixed test dataset are presented. This accuracy is not the highest but seems to be reasonably high, and the difference between the case of using the fixed datasets and that of introducing the cross-validation is substantially low. From the above, it is revealed that employing the method based on a combination of Sudachi, level 3 list, One Hot Encoding, and CatBoost learning is comprehensively favorable in constructing a discrimination model with high capability of the examination-category determination.

The scale of datasets treated in this paper is comparatively small, and hence the model constructed under the combination of Sudachi, level 3 list, One Hot Encoding, and CatBoost learning would be suited for ophthalmology clinics with difficulty of preparing a large-scale dataset. Let us discuss determining examination categories using NNs. In Tables 3 and 8, accuracies achieved by models that NN learning constructs are somewhat disappointing. It seems that low accuracies associated with NN models were caused by using the above small-scale dataset. The detailed evaluation should be made for the proposed method using NN learning with a large-scale dataset.

6 Conclusions

In this paper, a method of determining examination categories was proposed for ophthalmology patients. It depends on classification capabilities of models constructed by machine learning, and data presented to discrimination models are prepared from

sentences handwritten in medical questionnaires. The handwritten sentences must be transformed into numerical vectors. The proposed method chooses either MeCab or Sudachi to decompose character strings into parts of speech and to specify words characterizing symptoms and conditions of the patients. To assign values to elements corresponding to the specified words, the proposed method chooses one of the following means: One Hot Encoding, Bag of Words, and Word2Vec. For machine learning to the model construction, SVM, CatBoost, and NN are considered to be available. The proposed method tries to fix the combination of the above choices to acquire the powerful capability of determining the categories. The proposed method was evaluated using the five-fold cross-validation with 580 data. As a result, the discrimination model achieved 64.3% as the highest accuracy on average under the combination of Sudachi, One Hot Encoding, and CatBoost learning. The accuracy is then improved by 3% or more compared with the cases of conducting other methods. It was thus established that the proposed method based on the above combination copes well with the examination-category determination for comparatively small-scale datasets.

Outpatients visiting ophthalmology tends to handwrite words with similar meanings when they fill out their questionnaires. In this case, it seems that employing Word2Vec recently presented is preferable to employing Bag of Words producing simple vectors. This intuition, however, does not apply to determination of examination categories. It is therefore necessary to theoretically analyze this issue to improve the accuracy. Besides, introducing BERT [13] for vector preparation seems to be promising to realize powerful discrimination models. Ophthalmologists use not only questionnaires but also the results of visual acuity and intraocular pressure tests to accurately diagnose diseases. We will introduce the results of such tests as data for machine learning.

References

- [1] J. Uda, “The Change of the Action of Patients in Outpatient Departments by The Electronic Karte : A Study on the Effect of Medical Information System; Examining the Action of Patients in Outpatient Departments (Part.2),” (in Japanese) Summaries of technical papers of Annual Meeting, Architectural Institute of Japan, E-1, Architectural planning and design I, Building types and community facilities, planning and design method building construction system human factor studies planning and design theory, pp. 479-480, 2004.
- [2] T. Kato and Y. Uetsuka, “Considerations over the effectiveness of introducing electronic medical record system in shortening the waiting hours at out-patient department: comparison between the initial phase of the implementation and 20 months later,” (in Japanese) Journal of Tokyo Women’s Medical College, 80 (1 · 2), pp.9-13, 2010.

- [3] Y. Ji, Y. Yanagawa, and S. Miyazaki, “Reducing outpatient waiting times for hospital using queuing theory” (in Japanese). Journal of Japan Industrial Management Association, vol. 60, 6, pp. 297-305, 2010. .
- [4] T. Kudo, K. Yamamoto, and Y. Matsumoto, “Applying Conditional Random Fields to Japanese Morphological Analysis,” EMNLP 2004.
- [5] K. Takaoka, S. Hisamoto, N. Kawahara, M. Sakamoto, Y. Uchida, and Y. Matsumoto, “Sudachi: a Japanese Tokenizer for Business,” European Language Resources Association (ELRA), 2018.
- [6] S. Kawamura, S. Hisamoto, H. Manabe, K. Takaoka, Y. Uchida, T. Oka, and M. Asahara, “chiVe 2.0: Towards Practical Japanese Embedding with Sudachi and NWJC,” (in Japanese) Proceedings of the Twenty-sixth Annual Meeting of the Association for Natural Language Processing (NLP2020), pp.6-16, 2020.
- [7] R.E. Fan, P.H. Chen, and C.J. Lin, “Working set selection using second order information for training SVM.,” Journal of Machine Learning Research 6, pp.1889-1918, 2005.
- [8] A.V. Dorogush, V. Ershov, and A. Gulin, “CatBoost: gradient boosting with categorical features support,” Workshop on ML Systems at NIPS, 2017..
- [9] G.E. Hinton, “Connectionist learning procedures.” Artificial intelligence 40.1, pp.185-234, 1989.
- [10] N. Kamiura, A. Saitoh, T. Isokawa, N. Matsui, and H. Tabuchi, “Ophthalmological Examination Determination Using Data Classification Based on Support Vector Machines and Self-Organizing Maps,” Journal of Japan Society for Fuzzy Theory and Intelligent Informatics, Vol.26, No.2, pp.559-572, 2014. DOI:10.3156/jsoft.26.559.
- [11] S. Morita, N. Kamiura, T. Isokawa, T. Yumoto, A. Emura, T. Yamauchi, and H. Tabuchi, “Ophthalmological Examination Determination Using Data Classification Based on Feedforward Neural Networks,” Proc. 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2018. DOI: 10.1109/SMC.2018.00158.
- [12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” Proc. Proceedings of the International Conference on Learning Representations (ICLR), 2013.
- [13] J. Devlin, M.W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Proc. 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol 1, 2019.

CONSTRUCTION ALGORITHMS FOR TERNARY BENT FUNCTIONS DERIVED FROM THEIR PARTICULAR PROPERTIES

RADOMIR STANKOVIĆ

Mathematical Institute of SASA, 11000 Belgrade, Serbia

`radomir.stankovic@gmail.com`

MILENA STANKOVIĆ

University of Niš, Faculty of Electronic Engineering, 18 000 Niš, Serbia

`milena.stankovic@elfak.ni.ac.rs`

CLAUDIO MORAGA

Technical University of Dortmund, 44221 Dortmund, Germany

Technical University "Federico Santa María", Valparaíso, Chile

`claudio.moraga@tu-dortmund.de`

JAAKKO ASTOLA

Tampere University of Technology, Tampere, Finland

`Jaakko.Astola@outlook.com`

Abstract

Bent functions, either binary or multiple-valued, are mathematical objects attractive for studying since besides highest non-linearity as their primary characteristic, express some other interesting properties, some of which can be used to devise construction algorithms for bent functions. In particular, binary bent functions have a strictly specified number of non-zero values. In the same way, ternary bent functions satisfy certain requirements on the distribution of values of elements of their value vectors. These requirements can be used to specify six classes of ternary bent functions. Classes are mutually related by encoding of function values. Functions within a class are mutually related by permutation of elements in their function vectors. Given a basic ternary bent function, other

functions in the same class can be constructed by permutation matrices having a block structure similar to that of the factor matrices appearing in the Good-Thomas decomposition of the Cooley-Tukey Fast Fourier transform and related algorithms. Conversion of function vectors into matrices or equivalent matrix-valued vectors of smaller length leads to the redistribution of space complexity of related manipulation algorithms. In this matrix representations, construction of other bent functions from a given known bent function is performed by using either properties of bent functions or by manipulation of such representations in terms of FFT-like permutation matrices of dimensions smaller than the length of the function vector of the initial bent functions.

1 Introduction

There are different approaches in the study of bent functions and solving of many challenging tasks related to bent functions. Construction of bent functions is certainly among them, since it is important for possible applications. An approach to the construction of bent functions consist of manipulating known bent functions [?], [?], [?]. The considerations presented below belong to that area. In this context, reducing or redistributing the complexity of construction procedures is important to make them feasible in practice. The paper discloses an attempt towards that based on certain properties of bent functions and by using matrix manipulation of known bent functions.

2 Certain Peculiar Properties of Bent Functions

Boolean and multiple-valued functions are defined as mappings $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and $f : \{0, 1, \dots, p-1\}^n \rightarrow \{0, 1, \dots, p-1\}$, respectively. To provide methods and algorithms for handling such functions some algebraic structures are imposed on the domain and the range sets with suitably defined operations over their elements. Usually, the structure of a group is assumed for the domain and of a field for the range. Therefore, these functions are viewed as discrete functions defined on finite groups taking values in a field. In this way, powerful mathematical methods from signal processing can be applied to these functions. In particular, when viewed in this way, they can be processed by Fourier transforms on these groups [?]. Moreover, properties of the corresponding Fourier coefficients can be used in the definition or specification of certain classes of binary and multiple-valued functions, as is the case for example for bent functions, as will be discussed below.

In order to make the binary and multiple-valued functions compatible with basis functions in terms of which Fourier transforms are defined, the corresponding encod-

ing is performed when computing their Fourier spectra. It means, function values are encoded by the values that basis functions in Fourier representations can take, i.e., by the elements appearing in the corresponding Fourier transform matrices. For the Boolean and multiple-valued functions, these are the Walsh transform and the Vilenkin-Chrestenson transform, respectively [?]. In what follows, this is called complex encoding. For ternary functions, the elements of the Vilenkin-Chrestenson matrix are $1, e_1, e_2$, where $e_1 = -\frac{1}{2}(1 - i\sqrt{3})$, $e_2 = e_1^* = -\frac{1}{2}(1 + i\sqrt{3})$, and the complex encoding is the mapping $(0, 1, 2) \rightarrow (1, e_1, e_2)$. Further, the Hadamard ordering of basis functions in the Fourier transforms appears convenient since matches the structure of the assumed domain groups resulting in the Kronecker product representable transform matrices [?], [?], [?].

The restriction imposed on the range of function values results in corresponding restrictions in the spectral domain. This ensures that the inverse transform will produce a Boolean or multiple-valued function from the spectrum. For example, the maximal absolute value of a Walsh coefficient of a Boolean function is 2^n [?], [?]. This can be used as a feature defining linear Boolean functions. A Boolean function is linear if its Walsh spectrum has a single non-zero coefficient with absolute value 2^n , while all other coefficients are 0. Binary bent functions, defined as functions achieving maximal non-linearity, have a Walsh spectrum in which the maximal value of a coefficient is minimal and is equal $2^{n/2}$. Since all other coefficients have the same absolute value, the Walsh spectrum of a bent function is flat. Along with the requirement for maximal non-linearity, this feature is used to generalize the concept of bentness to ternary functions as well as other generalized bent functions in terms of the Fourier transform on the corresponding domain group [?], [?], [?], [?]. Therefore, an alternative definition of ternary bent functions is the following. A ternary function is bent if, after complex encoding, its Vilenkin-Chrestenson spectrum is flat, i.e., all the Vilenkin-Chrestenson coefficients have the absolute value $3^{n/2}$ [?].

This requirement for ternary bent functions in the spectral domain, imposes two properties in the original domain

1. Except for $n = 1$, all three values 0, 1, 2 must appear in the function vector of a function to be bent.
2. The number of appearances of different values in the function vector of a bent function is strictly specified as a triple $D = (d_0, d_1, d_2)$ called the distribution of function values.

Table ?? shows the distributions for ternary bent functions for $n = 1, 2, 3, 4, 5, 6, 7$ [?].

n	Distributions
1	$D = (0, 1, 2)$
2	$D = (5, 2, 2), D = (1, 4, 4)$
3	$D = (6, 9, 12)$
4	$D = (33, 24, 24), D = (21, 30, 30)$
5	$D = (72, 81, 90)$
6	$D = (225, 252, 252), D = (261, 234, 234)$
7	$D = (702, 729, 756)$

Table 1: Distribution of function values in ternary bent functions.

Notice that the distribution shows how many times different values appear in the function vector, but does not state which values repeat a given number d_i , $i \in \{0, 1, 2\}$, of times. Therefore, functions with the same distributions differ up to the encoding. If we determine the distribution for a given bent function, we get its composition $C = (c_0, c_1, c_2)$ which shows how many times each ternary value appears in its function vector. Functions with the same composition differ up to the permutation of values in their function vectors. In the present paper, we use this feature to construct several other bent functions from an initial bent function by using first properties of the Vilenkin-Chrestenson spectra as in Examples ?? and ??, and then permutation matrices preserving bentness [?], [?] in Examples ??, ??.

3 FFT-like Permutation Matrices

As noticed above, ternary bent functions with identical compositions have function vectors with permuted elements. It follows that some other bent functions can be constructed from a given bent function by permuting elements of its function vector. However, not every permutation of the function vector preserves the bentness, similarly as any combination of even integers can not be used as the Walsh spectrum of Boolean functions. It is a rather very restricted set of permutations that can be used, since the portion of bent functions out of all ternary functions for a given number of variables is very small. The restriction comes from the requirements in the original and spectral domain that should be simultaneously satisfied for a function to be bent. A set of allowed permutations is determined in [?] by referring to the spectral invariant operations since they preserve flatness of the spectrum, in other words, bentness of the functions.

In [?], four basic permutation matrices are defined, which map ternary functions in a single variable with the same composition to each other. These matrices are

$$\begin{aligned} \mathbf{Q}_1 &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Q}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \\ \mathbf{X}_1 &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{N}_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}. \end{aligned} \quad (1)$$

The basic permutation matrices for $n = 1$ permit constructing FFT-like permutation matrices for any larger n by an analogy with the application of basic transform matrices in defining factor matrices of the Good-Thomas factorisation of the Cooley-Tukey FFT [?].

Definition 1. *An FFT-like permutation matrix is defined as*

$$\mathbf{P}(n) = \bigotimes_{i=1}^n \mathbf{P}_j(1), \quad \mathbf{P}_j(1) = \begin{cases} \mathbf{q}(1) & \text{for } i = j, \\ \mathbf{I}(1) & \text{for } i \neq j, \end{cases} \quad (2)$$

where $\mathbf{q}(1)$ is any of the basic permutation matrices \mathbf{Q}_1 , \mathbf{Q}_2 , \mathbf{X}_1 , \mathbf{N}_1 , or \mathbf{X}_1^T , and $\mathbf{I}(1)$ is the (3×3) identity matrix.

Notice that except for \mathbf{X}_1 , which is orthogonal, the basic permutation matrices are self-transpose, i.e., they are symmetric and self-inverse.

Since FFT-like permutation matrices are derived from factor matrices in the FFT, it is ensured that they perform a permutation of spectral coefficients corresponding to certain spectral invariant operations. Therefore, their application ensures preserving bentness. The correspondence between these permutation matrices and spectral invariant operations is reviewed in [?]. For invariance of the spectrum with respect to permutations, we refer to [?].

4 Matrices Derived from Function Vectors

The function vector $\mathbf{F} = [f(0), f(1), \dots, f(3^n - 1)]^T$ specifies a ternary function in n variables by enumerating its values in all the points of the domain. Such vectors can be written as $(3^k \times 3^r)$ matrices \mathbf{Q} obtained by writing successive subvectors of length 3^k of \mathbf{F} as their rows, where $k = 1, 2, \dots, (n - 1)$. Different choices of parameters k and r lead to matrices of different dimensions.

This representation of bent functions is a basis for construction of bent functions due to the observations discussed below.

5 Construction of Ternary Bent Functions through Matrices

Transposition of a matrix means permuting of its row and column indices. If the matrix is build from the function vector of a bent function, then the transposition induces a permutation of the arguments of the function, which is a spectral invariant operation and ensures that bentness is preserved. From there, the following procedure immediately follows.

For a ternary bent function we represent the number of variables as $n = k + r$ where $k = 1, 2, \dots, (n - 1)$, and $r = n - k$. Then, we rewrite its function vector \mathbf{F} as $(3^k \times 3^r)$ matrices \mathbf{Q} for different values of k by writing subvectors of length 3^k as rows of \mathbf{Q} . If we take the transpose matrix of \mathbf{Q} for a value of k , and concatenate its rows, the obtained function is also bent. The reason is that the transposition corresponds to the permutation of variables defined as the cyclic shift for $(n - k)$ positions to the left.

Example 1. Consider a function of $n = 3$ variables. For simplicity, we write just indices of elements in its function vector

$$\mathbf{F} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, \\ 16, 1, 18, 19, 20, 21, 22, 23, 24, 25, 26]^T.$$

We write this function vector as a (9×3) matrix

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \\ 12 & 13 & 14 \\ 15 & 16 & 17 \\ 18 & 19 & 20 \\ 21 & 22 & 23 \\ 24 & 25 & 26 \end{bmatrix}.$$

The transpose (3×9) matrix is

$$\begin{bmatrix} 0 & 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \\ 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 & 25 \\ 2 & 5 & 8 & 11 & 14 & 17 & 20 & 23 & 26 \end{bmatrix}.$$

By concatenating the rows of this matrix, we obtain the function vector

$$\mathbf{F}_{new} = [0, 3, 6, 9, 12, 15, 18, 21, 24, 1, 4, 7, 10, 13, 16, 19, 22, 25, 2, 5, 8, 11, 14, 17, 20, 23, 26]^T$$

This is the vector which corresponds to the permutation of variables $x_1x_2x_3 \rightarrow x_2x_3x_1$.

If we first convert \mathbf{F} into a (3×9) matrix,

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 \end{bmatrix}$$

the transposition of it will produce the (9×3) matrix

$$\begin{bmatrix} 0 & 9 & 18 \\ 1 & 10 & 19 \\ 2 & 11 & 20 \\ 3 & 12 & 21 \\ 4 & 13 & 22 \\ 5 & 14 & 23 \\ 6 & 15 & 24 \\ 7 & 16 & 25 \\ 8 & 17 & 26 \end{bmatrix}.$$

Concatenation of rows of this matrix produces the vector

$$\mathbf{F}_{new} = [0, 9, 18, 1, 10, 19, 2, 11, 20, 3, 12, 21, 4, 13, 22, 5, 14, 23, 6, 15, 24, 7, 16, 25, 8, 17, 26]^T,$$

which corresponds to the permutation of variables $x_1x_2x_3 \rightarrow x_3x_1x_2$.

It follows that we obtain other bent functions for different values of k . The dimensions of rows and columns can be mutually exchanged which produces further possibilities for other bent functions. The method can be applied to bent functions in any number of variables and independently of their degree.

Example 2. Consider the bent function in four variables of degree three $f = x_1x_2 \oplus x_3x_4 \oplus x_1x_4^2 \oplus x_1x_3$. Its function vector is

$$\begin{aligned} \mathbf{F} = & [0, 0, 0, 0, 1, 2, 0, 2, 1 | 0, 0, 0, 0, 1, 2, 0, 2, 1 | 0, 0, 0, 0, 1, 2, 0, 2, 1 | \\ & 0, 1, 1, 1, 0, 1, 2, 2, 1 | 1, 2, 2, 2, 1, 2, 0, 0, 2 | 2, 0, 0, 0, 2, 0, 1, 1, 0 | \\ & 0, 2, 2, 2, 2, 0, 1, 2, 1 | 2, 1, 1, 1, 1, 2, 0, 1, 0 | 1, 0, 0, 0, 0, 1, 2, 0, 2]^T. \end{aligned}$$

For $k = 2$, it is $r = 2$ and we convert this function into the $(3^2 \times 3^2)$ matrix

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 1 & 2 & 0 & 0 & 2 \\ 2 & 0 & 0 & 0 & 2 & 0 & 1 & 1 & 0 \\ 0 & 2 & 2 & 2 & 2 & 0 & 1 & 2 & 1 \\ 2 & 1 & 1 & 1 & 1 & 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 \end{bmatrix}.$$

By concatenating the rows of \mathbf{Q}^T , we obtain the function vector

$$\begin{aligned} \mathbf{F}_1 = & [0, 0, 0, 0, 1, 2, 0, 2, 1 | 0, 0, 0, 1, 2, 0, 2, 1, 0 | 0, 0, 0, 1, 2, 0, 2, 1, 0 | \\ & 0, 0, 0, 1, 2, 0, 2, 1, 0 | 1, 1, 0, 1, 2, 2, 1, 0 | 2, 2, 2, 1, 2, 0, 0, 2, 1 | \\ & 0, 0, 0, 2, 0, 1, 1, 0, 2 | 2, 2, 2, 2, 0, 1, 2, 1, 0 | 1, 1, 1, 1, 2, 0, 1, 0, 2]^T \end{aligned}$$

of the function

$$f_1 = x_1x_2 \oplus x_3x_4 \oplus x_2^2x_3 \oplus x_1x_3,$$

which is also bent. This function corresponds to the cyclic shift of variables in f for $r = 2$ places to the left. Thus, $x_1, x_2, x_3, x_4 \rightarrow x_3, x_4, x_1, x_2$.

If the initial function f is converted into a (27×3) matrix, and its transpose determined, the produced bent function has the functional expression as

$$f_2 = x_2x_4 \oplus x_2x_3 \oplus x_1x_4 \oplus x_1^2x_2.$$

This function corresponds to the shift of variables in f for $r = 1$ places to the left.

The matrix (3×27) produces the function with the functional expression

$$f_3 = x_1x_4 \oplus x_2x_3 \oplus x_4x_3^2 \oplus x_2x_4.$$

This function corresponds to the shift of variables in f for $r = 3$ places to the left.

Example 3. Consider the function in 6 variables

$$f = 1 \oplus x_1x_2 \oplus x_3x_4 \oplus 2x_5x_6 \oplus x_2^2x_4x_5 \oplus x_6.$$

It is clear that this function is of degree 4, and by computing its Vilenkin-Chrestenson spectrum, after its complex encoding, it can be determined that it is bent.

The function vector is of length $3^6 = 729$, and can be converted into matrices of the dimensions (27×27) , (9×81) , (3×243) , (81×9) , (243×3) . After transposition and re-converting into function vectors, these matrices produce, respectively, five bent functions

$$\begin{aligned} f_1 &= 1 \oplus x_4x_5 \oplus x_3 \oplus 2x_2x_3 \oplus x_1x_6 \oplus x_1x_2x_5^2, \\ f_2 &= 1 \oplus x_5x_6 \oplus x_4 \oplus 2x_3x_4 \oplus x_2x_3x_6^2 \oplus x_1x_2, \\ f_3 &= 1 \oplus x_5 \oplus 2x_4x_5 \oplus x_2x_3 \oplus x_1x_6 \oplus x_1^2x_3x_4, \\ f_4 &= 1 \oplus x_5x_6 \oplus x_3x_4 \oplus x_2 \oplus x_1x_4^2x_6 \oplus 2x_1x_2, \\ f_5 &= 1 \oplus x_4x_5 \oplus x_3^2x_5x_6 \oplus x_2x_3 \oplus x_1 \oplus 2x_1x_6. \end{aligned}$$

These functions correspond to the cyclic shift of variables for $r = n - k$ places to the left. Therefore, the cyclic shift is for 3, 4, 5, 2, 1 places to the left. There are 3^n linear ternary functions to which any of two constants 1 and 2 can be added to get affine ternary functions. We can add $3^{n+1} = 3^7$ affine functions in 6 variables to any of these bent functions and in this way obtain more bent functions [?].

The following example illustrates that depending on the initial bent function, the number of produced bent functions is not necessarily equal to the number of possible matrices of different dimensions to be transposed. In certain cases, transposition of matrices of different dimensions might produce the same bent functions or functions equal to the initial functions.

Example 4. Consider the function represented as a sum of disjoint product of variables

$$f = x_1x_2 \oplus x_3x_4 \oplus x_5x_6.$$

This function is often considered as the basic bent function in 6 variables for the distribution $D = (225, 252, 252)$. The conversion into a (27×27) matrix and transposition of it produces

$$f_{\text{new},27} = x_4x_5 \oplus x_2x_3 \oplus x_1x_6.$$

The (9×81) matrix after transposition produces the initial function f . The (3×243) matrix after transposition produces the function equal to that obtained from the (27×27) matrix.

Example 5. The function represented by the sum of squares of variables

$$f = x_1^2 \oplus x_2^2 \oplus x_3^2 \oplus x_4^2 \oplus x_5^2 \oplus x_6^2.$$

is often considered as the basic bent function in 6 variables for the distribution $D = (261, 234, 234)$. Any of the three possible matrices after transposition produces the initial function f . This is due to the fact that f is symmetric.

The procedure to construct bent functions by manipulating a known bent functions used in the above examples can be formalized into an algorithm as follows.

- Algorithm 1.**
1. Given a ternary bent function in n variables specified as a function vector \mathbf{F} of length 3^n .
 2. Split \mathbf{F} into subvectors of length 3^k , for $k = 1, 2, \dots, (n - 1)$.
 3. Write the subvectors in Step 2 as rows of a $(3^k \times 3^{n-k})$ matrix \mathbf{Q} .
 4. Determine \mathbf{Q}^T , the transpose of \mathbf{Q} .
 5. Concatenate rows of \mathbf{Q}^T to produce the function vector of a bent function f_{new} .
 6. Check if the constructed function already exists in the list of obtained functions for different parameters k and r . If Yes, return to Step 2, if No add f_{new} to the list, and then return to Step 2.

As noticed above, in certain cases, depending on the patterns in the function vector of the initial function, the functions produced by the transposition for certain choices of k and r , might be identical to either the initial function or to the functions obtained for different choices of the parameters, as can be seen in Examples ?? and ?. In such cases, another approach presented in the next section based on matrix-valued equivalents of bent functions can be tried.

Transposition of matrices is an operation widely used in many computing and related application oriented algorithms and its implementation is well studied including hardware realizations. For an example of such circuits for transposition of matrices, we refer to Example 6.14 in [?].

The algorithm presented above immediately implies a challenging task of designing hardware for construction of ternary bent functions. If based on this algorithm, the required hardware reduces to a circuit performing the transposition of matrices. The circuit should be a reconfigurable circuit in order to accommodate different choices for the parameters k and r .

6 Matrix-valued Ternary Bent Functions

A possible approach towards redistribution of complexity of computing related to processing of large function vectors is to convert them into shorter matrix-valued

n	2	3	4	5	6	7	8
k	1	1	1, 2	1, 2	1, 2, 3	1, 2, 3	1, 2, 3, 4
r	1	3	9, 1	27, 3	81, 9, 1	243, 27, 3	739, 81, 9, 1

Table 2: Possible decompositions of function vectors into matrix-valued elements.

vectors [?]. In this way, the representation of functions and required computing with large vectors is converted into representations and computing with vectors of smaller length and their matrix-valued elements. The size of vectors and matrices used as their elements can be varied depending on concrete applications and available computing resources.

In [?], matrix-valued equivalents of ternary bent functions are defined and used for classification of bent functions in terms of patterns appearing in the corresponding matrix-valued coefficients. In this section, we use the matrix-valued equivalents of bent functions as a basis to construct bent functions from the given bent functions. We split the function vector into subvectors of length 3^k , $k = 1, 2, \dots, \lceil (n-1)/2 \rceil$, where $\lceil a \rceil$ is the smallest integer greater or equal to a , and write them as $(3^k \times 3^k)$ matrices. In this way, the function vector \mathbf{F} of length 3^n is converted into a matrix-valued equivalent function vector of $r = 3^{n-2k}$ elements. A function vector is reconstructed from the matrix-valued equivalent by concatenating the rows of its elements starting from the first element and by processing elements successively. It means that we first concatenate the rows of the first element, then the rows of the second element, and continue in the same way.

Table ?? shows the possible dimensions and the number of matrix-valued elements for $n = 2, 3, 4, 5, 6, 7, 8$. For simplicity, the considerations and examples in this section are given for $k = 1$, i.e., the matrix-valued elements of function vectors are (3×3) matrices. All the statements and observations are valid for other possible decompositions.

Remark 1. *For a matrix-valued equivalent of a bent function for all the choices of parameters k and r , the determinant of matrix-valued elements is equal to 0.*

Example 6. *Consider function $f_{1324}(x_1, x_2, x_3, x_4) = x_1x_3 \oplus x_2x_4$ and write its function vector into the form of a vector of 9 elements whose entries are (3×3) matrices obtained by arranging triples of successive elements into rows of these matrices. Thus, this function is represented as*

$$\mathbf{F}_{1324} = [\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_5, \mathbf{a}_6, \mathbf{a}_7, \mathbf{a}_8]^T,$$

where

$$\begin{aligned} \mathbf{a}_0 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \mathbf{a}_1 &= \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix} & \mathbf{a}_2 &= \begin{bmatrix} 0 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix} \\ \mathbf{a}_3 &= \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix} & \mathbf{a}_4 &= \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix} & \mathbf{a}_5 &= \begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{bmatrix} \\ \mathbf{a}_6 &= \begin{bmatrix} 0 & 0 & 0 \\ 2 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix} & \mathbf{a}_7 &= \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix} & \mathbf{a}_8 &= \begin{bmatrix} 0 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 2 \end{bmatrix}. \end{aligned}$$

We determine a matrix-valued function with elements obtained as $q_i = \mathbf{X}_1 \mathbf{a}_i$, where \mathbf{X}_1 is the basic permutation matrix. Therefore,

$$\mathbf{F}_{132412} = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5, \mathbf{v}_6, \mathbf{v}_7, \mathbf{v}_8]^T$$

where

$$\begin{aligned} \mathbf{v}_0 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \mathbf{v}_1 &= \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix} & \mathbf{v}_2 &= \begin{bmatrix} 0 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix} \\ \mathbf{v}_3 &= \begin{bmatrix} 2 & 2 & 2 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} & \mathbf{v}_4 &= \begin{bmatrix} 2 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix} & \mathbf{v}_5 &= \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix} \\ \mathbf{v}_6 &= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 2 & 2 & 2 \end{bmatrix} & \mathbf{v}_7 &= \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix} & \mathbf{v}_8 &= \begin{bmatrix} 1 & 0 & 2 \\ 0 & 2 & 1 \\ 2 & 1 & 0 \end{bmatrix}. \end{aligned}$$

This matrix-valued function defines, after concatenating the rows of matrix-valued elements, a function with the function vector

$$\begin{aligned} \mathbf{F}_{12} &= [0, 0, 0, 0, 0, 0, 0, 0, 0 | 0, 1, 2, 0, 1, 2, 0, 1, 2 | 0, 2, 1, 0, 2, 1, 0, 2, 1 | \\ &\quad 2, 2, 2, 0, 0, 0, 1, 1, 1 | 2, 0, 1, 0, 1, 2, 1, 2, 0 | 2, 1, 0, 0, 2, 1, 1, 0, 2 | \\ &\quad 1, 1, 1, 0, 0, 0, 2, 2, 2 | 1, 2, 0, 0, 1, 2, 2, 0, 1 | 1, 0, 2, 0, 2, 1, 2, 1, 0]^T \end{aligned}$$

and whose functional expression is

$$f_{12} = f_{1324}(x_1, x_2, x_3, x_4) \oplus 2x_1 = x_1x_3 \oplus x_2x_4 \oplus 2x_1.$$

This function is bent since the linear term $2x_1$ is added to f_{1324} .

We now construct another function the elements of which are obtained from the corresponding coefficients in the initial function as $s_i = \mathbf{X}_1^T \mathbf{a}_i$, Therefore,

$$\mathbf{F}_{1324122} = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5, \mathbf{s}_6, \mathbf{s}_7, \mathbf{s}_8]^T,$$

where

$$\begin{aligned} \mathbf{s}_0 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \mathbf{s}_1 &= \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix} & \mathbf{s}_2 &= \begin{bmatrix} 0 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix} \\ \mathbf{s}_3 &= \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 0 & 0 & 0 \end{bmatrix} & \mathbf{s}_4 &= \begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} & \mathbf{s}_5 &= \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \\ \mathbf{s}_6 &= \begin{bmatrix} 2 & 2 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} & \mathbf{s}_7 &= \begin{bmatrix} 2 & 0 & 1 \\ 1 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix} & \mathbf{s}_8 &= \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & 2 \\ 0 & 2 & 1 \end{bmatrix}. \end{aligned}$$

From matrix-valued coefficients, we derive the function vector of another bent function whose function vector is

$$\begin{aligned} \mathbf{F}_{1324122} &= [0, 0, 0, 0, 0, 0, 0, 0, 0 | 0, 1, 2, 0, 1, 2, 0, 1, 2 | 0, 2, 1, 0, 2, 1, 0, 2, 1 | \\ &\quad 1, 1, 1, 2, 2, 2, 0, 0, 0 | 1, 2, 0, 2, 0, 1, 0, 1, 2 | 1, 0, 2, 2, 1, 0, 0, 2, 1 | \\ &\quad 2, 2, 2, 1, 1, 1, 0, 0, 0 | 2, 0, 1, 1, 2, 0, 0, 1, 2 | 2, 1, 0, 1, 0, 2, 0, 2, 1]^T. \end{aligned}$$

This function is bent which can be seen from its Vilenkin-Chestenson spectrum, and its functional expression is

$$f_{1324122} = f_{1234}(x_1, x_2, x_3, x_4) = x_1x_3 \oplus x_2x_4 \oplus 2x_1x_2^2.$$

In this case, the term $2x_1x_2^2$ is added.

Example 7. In the present example, we first label elements of the matrix-valued function derived from the initial function by 0, 1, 2, 3, 4, 5, 6, 7, 8 and obtain a function vector $\mathbf{F}_{1324,e} = [0, 1, 2, 3, 4, 5, 6, 7, 8]^T$. We permute it by the permutation matrix $\mathbf{P}_{x,i} = \mathbf{X}_1 \otimes \mathbf{I}(1)$ defined in (??) [?] and produce a function vector $\mathbf{F}_{x,i,q,1324,e} =$

$[6, 7, 8, 0, 1, 2, 3, 4, 5]^T$. The replacement of entries in this vector by the corresponding matrix-valued elements of \mathbf{F}_{1324} produces the matrix-valued function as follows

$$\mathbf{F}_{x,i,q,1324,e} = [\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4, \mathbf{w}_5, \mathbf{w}_6, \mathbf{w}_7, \mathbf{w}_8]^T,$$

where

$$\begin{aligned} \mathbf{w}_0 &= \begin{bmatrix} 0 & 0 & 0 \\ 2 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix} & \mathbf{w}_1 &= \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix} & \mathbf{w}_2 &= \begin{bmatrix} 0 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 2 \end{bmatrix} \\ \mathbf{w}_3 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \mathbf{w}_4 &= \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix} & \mathbf{w}_5 &= \begin{bmatrix} 0 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix} \\ \mathbf{w}_6 &= \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix} & \mathbf{w}_7 &= \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix} & \mathbf{w}_8 &= \begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{bmatrix}. \end{aligned}$$

Conversion into a function vector produces a function that is bent and its function vector is

$$\begin{aligned} \mathbf{F}_{x,i,q,1324} &= [0, 0, 0, 2, 2, 2, 1, 1, 1 | 0, 1, 2, 2, 0, 1, 1, 2, 0 | 0, 2, 1, 2, 1, 0, 1, 0, 2 | \\ &\quad 0, 0, 0, 0, 0, 0, 0, 0, 0 | 0, 1, 2, 0, 1, 2, 0, 1, 2 | 0, 2, 1, 0, 2, 1, 0, 2, 1 | \\ &\quad 0, 0, 0, 1, 1, 1, 2, 2, 2 | 0, 1, 2, 1, 2, 0, 2, 0, 1 | 0, 2, 1, 1, 0, 2, 2, 1, 0]^T. \end{aligned}$$

This function is bent and has the functional expression as

$$f_{x,i,q,1324} = x_1x_3 \oplus x_2x_4 \oplus 2x_3.$$

If the encoded function vector $\mathbf{F}_{1324,e}$ is reordered by the permutation $\mathbf{P}_{i,x} = \mathbf{I}(1) \otimes \mathbf{X}_1$, we get the function vector $\mathbf{F}_{i,x,q,1324,e} = [2, 0, 1, 5, 3, 4, 8, 6, 7]^T$. The replacement of entries by the matrix-valued elements of \mathbf{F}_{1324} produces

$$\mathbf{F}_{e,i,x,q,1324} = [\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_7, \mathbf{d}_8]^T,$$

where

$$\begin{aligned} \mathbf{d}_0 &= \begin{bmatrix} 0 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix} & \mathbf{d}_1 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \mathbf{d}_2 &= \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix} \\ \mathbf{d}_3 &= \begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{bmatrix} & \mathbf{d}_4 &= \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix} & \mathbf{d}_5 &= \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix} \\ \mathbf{d}_6 &= \begin{bmatrix} 0 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 2 \end{bmatrix} & \mathbf{d}_7 &= \begin{bmatrix} 0 & 0 & 0 \\ 2 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix} & \mathbf{d}_8 &= \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}. \end{aligned}$$

Conversion into a function vector produces

$$\begin{aligned} \mathbf{F}_{i,x,q,1324} &= [0, 2, 1, 0, 2, 1, 0, 2, 1 | 0, 0, 0, 0, 0, 0, 0, 0 | 0, 1, 2, 0, 1, 2, 0, 1, 2 | \\ &\quad 0, 2, 1, 1, 0, 2, 2, 1, 0 | 0, 0, 0, 1, 1, 1, 2, 2 | 2, 0, 1, 2, 1, 2, 0, 2, 0, 1 | \\ &\quad 0, 2, 1, 2, 1, 0, 1, 0, 2 | 0, 0, 0, 2, 2, 2, 1, 1, 1 | 0, 1, 2, 2, 0, 1, 1, 2, 0]^T. \end{aligned}$$

This is a function which is bent and has the functional expression as

$$f_{e,i,x,q,1324} = x_1x_3 \oplus x_2x_4 \oplus 2x_4.$$

If the entries of $\mathbf{F}_{i,x,q,1324} = [2, 0, 1, 5, 3, 4, 8, 6, 7]^T$ are replaced by the matrix-valued elements of \mathbf{F}_{132412} , thus, by elements of \mathbf{F}_{1324} permuted by \mathbf{X}_1 , we get

$$\mathbf{F}_{i,x,q,132412,e} = [\mathbf{h}_0, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}_4, \mathbf{h}_5, \mathbf{h}_6, \mathbf{h}_7, \mathbf{h}_8]^T,$$

where

$$\begin{aligned} \mathbf{h}_0 &= \begin{bmatrix} 0 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix} & \mathbf{h}_1 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \mathbf{h}_2 &= \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix} \\ \mathbf{h}_3 &= \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix} & \mathbf{h}_4 &= \begin{bmatrix} 2 & 2 & 2 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} & \mathbf{h}_5 &= \begin{bmatrix} 2 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix} \\ \mathbf{h}_6 &= \begin{bmatrix} 1 & 0 & 2 \\ 0 & 2 & 1 \\ 2 & 1 & 0 \end{bmatrix} & \mathbf{h}_7 &= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 2 & 2 & 2 \end{bmatrix} & \mathbf{h}_8 &= \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}. \end{aligned}$$

Conversion into a function vector produces

$$\begin{aligned} \mathbf{F}_{i,x,q,132412} &= [0, 2, 1, 0, 2, 1, 0, 2, 1 | 0, 0, 0, 0, 0, 0, 0, 0 | 0, 1, 2, 0, 1, 2, 0, 1, 2 | \\ &\quad 2, 1, 0, 0, 2, 1, 1, 0, 2 | 2, 2, 2, 0, 0, 0, 1, 1, 1 | 2, 0, 1, 0, 1, 2, 1, 2, 0 | \\ &\quad 1, 0, 2, 0, 2, 1, 2, 1, 0 | 1, 1, 1, 0, 0, 0, 2, 2, 2 | 1, 2, 0, 0, 1, 2, 2, 0, 1]^T, \end{aligned}$$

This function is bent and has the functional expression

$$f_{i,x,q,132412} = x_1x_3 \oplus x_2x_4 \oplus 2x_1 \oplus 2x_4.$$

This function is bent since the linear term $2x_1 \oplus 2x_4$ is added.

Example 8. *Consider the function $f_{\text{square}} = x_1^2 \oplus x_2^2 \oplus x_3^2 \oplus x_4^2$. Its function vector is*

$$\begin{aligned} \mathbf{F}_{\text{square}} = & [0, 1, 1, 1, 2, 2, 1, 2, 2 | 1, 2, 2, 2, 0, 0, 2, 0, 0 | 1, 2, 2, 2, 0, 0, 2, 0, 0 | \\ & 1, 2, 2, 2, 0, 0, 2, 0, 0 | 2, 0, 0, 0, 1, 1, 0, 1, 1 | 2, 0, 0, 0, 1, 1, 0, 1, 1 | \\ & 1, 2, 2, 2, 0, 0, 2, 0, 0 | 2, 0, 0, 0, 1, 1, 0, 1, 1 | 2, 0, 0, 0, 1, 1, 0, 1, 1]^T. \end{aligned}$$

The corresponding matrix-valued function is

$$\mathbf{F}_{\text{square}} = [\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4, \mathbf{r}_5, \mathbf{r}_6, \mathbf{r}_7, \mathbf{r}_8]^T,$$

where

$$\begin{aligned} \mathbf{r}_0 &= \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 2 \end{bmatrix} & \mathbf{r}_1 &= \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} & \mathbf{r}_2 &= \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \\ \mathbf{r}_3 &= \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} & \mathbf{r}_4 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{r}_5 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \\ \mathbf{r}_6 &= \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} & \mathbf{r}_7 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{r}_8 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \end{aligned}$$

We first perform the encoding of matrix-valued elements of $\mathbf{F}_{\text{square}}$ as $\mathbf{F}_{e,\text{square}} = [0, 1, 2, 3, 4, 5, 6, 7, 8]^T$. Then, we apply the FFT-like permutation matrix $\mathbf{Q}_{i,x^T} = \mathbf{I}(1) \otimes \mathbf{X}_1^T$. The reordered function is $\mathbf{F}_{e,\text{square},\text{new}} = [1, 2, 0, 4, 5, 3, 7, 8, 6]^T$. We re-assign the matrix-valued elements to the reordered function as

$$\mathbf{F}_{e,\text{square},\text{new}} = [\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4, \mathbf{y}_5, \mathbf{y}_6, \mathbf{y}_7, \mathbf{y}_8]^T,$$

where

$$\begin{aligned}
 \mathbf{y}_0 &= \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} & \mathbf{y}_1 &= \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} & \mathbf{y}_2 &= \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 2 \end{bmatrix} \\
 \mathbf{y}_3 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{y}_4 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{y}_5 &= \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \\
 \mathbf{y}_6 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{y}_7 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{y}_8 &= \begin{bmatrix} 1 & 2 & 2 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}.
 \end{aligned}$$

When converted into a vector, we obtain the function vector

$$\begin{aligned}
 \mathbf{F}_{\text{square}} &= [1, 2, 2, 2, 0, 0, 2, 0, 0 | 1, 2, 2, 2, 0, 0, 2, 0, 0 | 0, 1, 1, 1, 2, 2, 1, 2, 2 | \\
 &\quad 2, 0, 0, 0, 1, 1, 0, 1, 1 | 2, 0, 0, 0, 1, 1, 0, 1, 1 | 1, 2, 2, 2, 0, 0, 2, 0, 0 | \\
 &\quad 2, 0, 0, 0, 1, 1, 0, 1, 1, 2, 0, 0, 0, 1, 1, 0, 1, 1 | 1, 2, 2, 2, 0, 0, 2, 0, 0]^T.
 \end{aligned}$$

of a bent function whose functional expression is

$$f = 1 \oplus 2x_2 \oplus x_1^2 \oplus x_2^2 \oplus x_3^2 \oplus x_4^2.$$

If we permute the matrix-valued elements in the reordered vector $\mathbf{F}_{e,\text{square},\text{new}}$ by any of the elementary permutation matrices, some other bent functions are constructed. For example, by \mathbf{Q}_1 , we obtain another bent function whose matrix-valued function vector is

$$\mathbf{F}_{\text{square},\text{new},\text{permuted}} = [\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4, \mathbf{g}_5, \mathbf{g}_6, \mathbf{g}_7, \mathbf{g}_8]^T,$$

where

$$\begin{aligned}
 \mathbf{g}_0 &= \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 2 \\ 2 & 0 & 0 \end{bmatrix} & \mathbf{g}_1 &= \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 2 \\ 2 & 0 & 0 \end{bmatrix} & \mathbf{g}_2 &= \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 1 \\ 1 & 2 & 2 \end{bmatrix} \\
 \mathbf{g}_3 &= \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{g}_4 &= \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{g}_5 &= \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 2 \\ 2 & 0 & 0 \end{bmatrix} \\
 \mathbf{g}_6 &= \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{g}_7 &= \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} & \mathbf{g}_8 &= \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 2 \\ 2 & 0 & 0 \end{bmatrix}.
 \end{aligned}$$

When expanded, we obtain the function vector of a bent function whose function vector is

$$\begin{aligned} \mathbf{F}_{\text{square}} = & [2, 0, 0, 1, 2, 2, 2, 0, 0 | 2, 0, 0, 1, 2, 2, 2, 0, 0 | 1, 2, 2, 0, 1, 1, 1, 2, 2 | \\ & 0, 1, 1, 2, 0, 0, 0, 1, 1 | 0, 1, 1, 2, 0, 0, 0, 1, 1 | 2, 0, 0, 1, 2, 2, 2, 0, 0 | \\ & 0, 1, 1, 2, 0, 0, 0, 1, 1 | 0, 1, 1, 2, 0, 0, 0, 1, 1 | 2, 0, 0, 1, 2, 2, 2, 0, 0]^T. \end{aligned}$$

and the corresponding functional expression is

$$f = 2 \oplus 2x_2 \oplus x_3 \oplus x_1^2 \oplus x_2^2 \oplus x_3^2 \oplus x_4^2.$$

This example illustrates that unlike the Algorithm ??, the method based on matrix-valued equivalents produces different bent functions also in the case of functions represented by the sum of squares of variables.

The permutation of matrix-valued elements can be performed by any of the basic transform matrices. Moreover, splitting into matrix-valued coefficients of different dimensions can be done, and in this case, FFT-like permutation matrices of the corresponding dimensions are used. In this way, certain other bent functions are obtained for different dimensions of matrix-valued coefficients.

The procedure presented in the above examples, can be summarised in the following algorithm.

- Algorithm 2.** 1. Given a ternary bent function in n variables by its function vector \mathbf{F} of length 3^n .
2. Split \mathbf{F} into subvectors of length 3^k , for $k = 1, 2, \dots, \lceil n/2 \rceil$, and write as a matrix-valued equivalent \mathbf{F}_{mv} with $r = 3^{n-2k}$ elements which are $(3^k \times 3^k)$ matrices.
3. Do encoding of elements of \mathbf{F}_{mv} by integers $0, 1, \dots, r-1$ and produce $\mathbf{F}_{mv,e}$.
4. Permute the encoded vector $\mathbf{F}_{mv,e}$ by an $(r \times r)$ permutation matrix \mathbf{P} defined in (??), i.e., by $\mathbf{P}(n-2k)$.
5. Re-assign to elements of $\mathbf{F}_{mv,e}$ the corresponding matrix-valued elements of \mathbf{F}_{mv} .
6. Concatenate the rows of matrix-valued elements of $\mathbf{F}_{mv,e}$ into the function vector of a bent function \mathbf{f}_{new} .
7. Permute each matrix-valued element of \mathbf{F}_{mv} by the same $(3^k \times 3^k)$ permutation matrix defined in (??).

8. Repeat the Steps 5, 6, and 7.
9. Repeat the Step 4 for a different permutation matrix, and then repeat Steps 5, 6, and 7.

7 Closing Remarks

An alternative definition of ternary bent functions states that these are functions with flat Vilenkin-Chrestenson spectra meaning that all the spectral coefficients take the same absolute value equal to $3^{n/2}$, where n is the number of variables. Therefore, in constructing bent functions, preserving bentness reads as preserving flatness of the spectrum. Thus, operations allowed in manipulating function vectors of bent functions reduce to the so-called spectral invariant operations, which permute the spectral coefficients or change their sign, but not their absolute values.

Permutations in the spectral domain, i.e., permutations of spectral coefficients, can be equivalently expressed as certain permutations of function values in the original domain. Some of the possible permutations are expressed by the FFT-like permutation matrices introduced for binary and ternary functions in [?], and [?], respectively.

We point out that changing the phase of the Vilenkin-Chrestenson coefficients for bent functions, which preserves their bentness, can be expressed in terms of transposition of matrices derived from their function vectors. In this way, bent functions can be constructed by matrix transposition which in practical implementation reduces to reading of function values in a particular exactly determined order. The space complexity of the procedure is proportional to the space required to store the known initial bent functions for the procedure. Further, this approach paves a way for the usage of the hardware for matrix transposition to construct bent functions.

Another approach to the redistribution and due to this possibly also reduction of space complexity of the construction procedure is through conversion of known bent functions into matrix-valued equivalents. In this way, a function vector of length 3^n is converted into a considerably shorter matrix-valued vector. The length of this vector and the dimension of matrix-valued coefficients can be chosen in different ways leading to different bent functions. These functions are obtained by permuting either or both reduced size matrix-valued function vectors and matrix-valued coefficients by FFT-like permutation matrices introduced for binary and ternary functions in [?], and [?], respectively. Matrix based construction algorithms are useful since they produce bent functions as sequences, which is more convenient for their possible applications instead of functional expressions.

We believe that algorithms presented above raise some tasks for further research works. First, devising reconfigurable hardware for construction of bent functions by starting from the hardware for transposition of matrices seems as a challenging task. Then, matrix based algorithms are in general suitable for the implementation on the contemporary GPU hardware. Implementation of the proposed algorithms may lead to interesting programming tasks.

References

- [1] Astola, J. T., Stanković, R. S., *Fundamentals of Switching Theory and Logic Design*, Springer, 2006.
- [2] Carlet, C., Mesnager, S., "Four decades of research on bent functions", *DES. Codes Cryptogr.*, 78, 2016, 5-50.
- [3] Hodžić, S., Pašalić, E., "Generalized bent functions - some general construction methods and related necessary and sufficient conditions", *Cryptogr. Commun.*, 2015, DOI 10.1007/s12095-015-0126-9.
- [4] Hurst, S. L., *Logical Processing of Digital Signals*, Crane Russak and Edward Arnold, London and Basel, 1978.
- [5] Hurst, S. L., Miller, D. M., Muzio, J. C., *Spectral Techniques for Digital Logic*, Academic Press, 1985.
- [6] Kumar, P. V., Scholtz, R. A., Welch, L. R., "Generalized bent functions and their properties", *J. Combin. Theory Ser. A*, Vol. 40, 1985, 90-107.
- [7] Karpovsky, M. G., Stanković, R. S., Astola, J., *Spectral Logic and its Applications for the Design of Digital Devices*, Wiley, 2008.
- [8] Luis, M., Moraga, C., "Functions with flat Chrestenson spectra", *Proc. 19th Int. Symp. Multiple-valued Logic*, Guangzhou, China, 1989, 406-413.
- [9] Luo, G., Cao, X., Mesnager, S., "Several new classes of self-dual bent functions derived from involutions", *Cryptography and Communications*, Published online 17 May 2019, Springer Science+Business Media, <https://doi.org/10.1007/s12095-019-00371-9>.
- [10] Mesnager, S., "On constructions of bent functions from involutions", *Proc. of Int. Symp. on Inform. Theory (ISIT)*, Barcelona, Spain, July 10-15, 2016, 110-114.
- [11] Moraga, C., "Permutations under spectral transforms", *Proc. 38th Int. Symp. on Multiple-valued Logic*, Dallas, Texas, USA, May 22-24, 2008, 76-81.
- [12] Stanković, R. S., Astola, J., "Design of decision diagrams with increased functionality of nodes through group theory", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2003, Vol. E86, No. A(3), 693-703.
- [13] Stanković, R. S., Stanković, M. M., Astola, J. T., Moraga, C., "Remarks on similarities among ternary bent functions", in *Proc. 49th Int. Symp. Multiple-valued Logic*, Fredericton NB, Canada, 2019, IEEE Press, 79-84.

- [14] Stanković, R. S., Stanković, M., Moraga, C., Astola, J. T., "Construction of binary bent functions by FFT-like permutation algorithms", *Proc. 14th Int. Workshop on Boolean Problems*, Bremen, Germany, September 24-25, 2020.
- [15] Stanković, R. S., Stanković, M., Moraga, C., Astola, J. T., "Construction of ternary bent functions by FFT-like permutation algorithms", *Proc. 50th Int. Symp on Multiple-valued Logic*, Miyazaki, Japan, November 9-11, 2020.
- [16] Tokareva, N., "Generalizations of bent functions - A survey", *J. Appl. Ind. Math.*, Vol. 5, No. 1, 2011, 110-129.
- [17] Tokareva, N., *Bent Functions - Results and Applications to Cryptography*, Elsevier, 2015.

QUANTUM MACHINE LEARNING, LOGIC MINIMIZATION, AND CIRCUIT DESIGN BY OPTIMIZING TERNARY-INPUT BINARY-OUTPUT KRONECKER REED-MULLER FORMS

MAGGIE BAO

Department of Electrical and Computer Engineering, Portland State University
maggie.bao202@gmail.com

COLE POWERS

Department of Electrical and Computer Engineering, Portland State University
colehpowers@gmail.com

MAREK PERKOWSKI

Department of Electrical and Computer Engineering, Portland State University
h8mp@pdx.edu

Abstract

This paper introduces a new spectral transform for ternary-input binary-output functions that generalizes the well-known binary Kronecker Reed-Muller forms. The two binary Davio Expansions are generalized to 27 ternary-input Davio Expansions. The new Hybrid Kronecker Reed-Muller spectrum has 28^n expansions for n ternary variables. By creating an oracle for this problem, we generalize past quantum Grover-based algorithms presented for the binary Fixed-Polarity Reed-Muller and Kronecker Reed-Muller transforms. Our quantum algorithm is for different variants of ternary-input binary-output forms realized on Grover's Algorithms with either binary or binary/ternary control variables. We create several variants of expansions including binary and ternary control variables, which leads to a modification of Grover's Algorithm for term minimization in the new expressions. The algorithm can be applied to incompletely specified functions, thus, introducing a new approach to quantum Machine Learning. The results of the binary Grover's Algorithm simulation in Qiskit are presented.

1 Introduction

The need to find the minimal expression of a certain function commonly arises in both circuit minimization and in some Machine Learning (ML) approaches [4,12,22]. Several ML problems can be formulated as the minimization of expressions for an unspecified Multiple-Valued (MV) function, binary being a special case.

L. Li, M. Thornton, and M. Perkowski [15] used Grover's Search Algorithm to create a quantum algorithm using a quantum circuit oracle to minimize polynomial expressions for binary Boolean functions in the family of all Fixed-Polarity Reed-Muller (FPRM) forms. The algorithm gave a quadratic complexity speedup with respect to the classical algorithm and selected the least expensive expansion form (expression) out of 2^n forms for a function of n variables. B. Lee and M. Perkowski [13] have expanded [15] to Grover's Algorithm with ternary control variables and the binary Kronecker Reed-Muller (KRM) expansions for incompletely specified binary data characteristic for Machine Learning. The best KRM form was selected out of 3^n forms with quadratic speedup based on quantum parallelism inherent to Grover's Algorithm. Additionally, [13] applied these methods to the learning of a humanoid robot HR-OS1.

The ternary-input binary-output forms that generalize FPRM and KRM, the simplest possible generalization of the Reed-Muller family of transforms to multi-valued logic, were not discussed in the literature. The goal of this paper is to do the same as [15] and [13] has done for ternary-input, binary-output functions, but for completely specified and incompletely specified ternary-input, binary-output functions. We call these new forms the Hybrid Kronecker Reed-Muller Expansions. The total number of these new forms is 28^n for ternary-input functions of n variables (the proof is given in Section 3.3.1). This creates a complicated kernel of butterfly, therefore, we also created algorithm variants for some smaller subsets of expansions. The binary-encoded variant selects $2^3 = 8$ expansions out of 28 expansions. These expansions were selected for their simplest circuits. Similarly, the ternary-encoded variant is for two variables and it has $3^2 = 9$ expansions. For each variable; the ternary variant includes 9 expansions.

2 Machine learning, Reed-Muller expansions, and butterfly circuits

This section reviews the minimum necessary background on ML and binary logic expansions used in reversible circuits and quantum computing.

2.1 Machine Learning

One approach to ML, emphasized, for instance, in Sum-of-Products Minimization [8], Exclusive-Sum-of-Products Minimization [23], Ashenhurst-Curtis Decomposition [16], Rule Based Extraction (AQ of Michalski [18] and CN2 of Clark and Niblett [5]), and Rough Set Approaches [14], is based on function minimization and number of variables minimization [19] to satisfy the logical principle, Occam's Razor. For instance, an ML classifier design problem can be formulated as minimizing a highly unspecified discrete function, often with multi-valued input but with binary output. Occam's Razor is satisfied by minimizing the complexity of the circuit. The input vectors of a multiple-input, single-output function represent samples of the data set, while the output of either a 0 or 1 represents the classification of the data into two categories based on the input features in a supervised type of learning. Given features of the data and knowledge of how some of the data are classified (supervised learning), the ML problem is to find the simplest Boolean expression that encompasses the training data, which is then used to classify new data (the testing phase of validation). Thus, this expression becomes a classifier, realized in software or in hardware. When the expression is realized as a reversible circuit with permutative (reversible) gates such as the generalized Toffoli, it becomes an oracle. Measurement gates are added on outputs. Thus, this oracle, trained on binary data, can produce learned results on arbitrary initializing quantum states instead of only basic binary states of Hilbert Space.

Don't-cares in logic design are called *don't-knows* in ML. Learning consists of converting the don't-knows to care values of 0s or 1s. When multiple completely satisfied learned functions meet the requirement of correct classification, by Occam's Razor, the simplest one will likely make the best predictions with which to replace the don't-knows of the initial specification. This paper describes a method of finding such a minimal expression given the set of samples (ternary minterms) with labels 0s and 1s in the learning data set.

2.2 Binary logic expansions

A binary function can be described by its true minterms (1s) and false minterms (0s). Any binary function with a single output can be written as a canonical *Exclusive-OR Sum-of-Products* (ESOP) of minterms. A binary function of three variables is given in Equation (1),

$$f(x_1, x_2, x_3) = m_0\bar{x}_1\bar{x}_2\bar{x}_3 \oplus m_1\bar{x}_1\bar{x}_2x_3 \oplus m_2\bar{x}_1x_2\bar{x}_3 \oplus m_3\bar{x}_1x_2x_3 \oplus m_4x_1\bar{x}_2\bar{x}_3 \oplus m_5x_1\bar{x}_2x_3 \oplus m_6x_1x_2\bar{x}_3 \oplus m_7x_1x_2x_3 \quad (1)$$

where $x_i \in \{0, 1\}$ are binary input variables of the completely specified function, the bar denotes the NOT operator, and $m_i \in \{0, 1\}$ are the minterms of the function. Again, this expansion is canonical, and it is useful only as the initial specification of the minimization problem. The corresponding general model of the classifier is an EXOR of arbitrary products of literals, called a general ESOP.

In binary logic, *literals* are variables or negated variables, wherein it being positive or negative denotes the *polarity*. Thus, for every canonical ESOP we can find ESOPs with smaller costs by modifying variable polarities and applying the ESOP tautology equivalence rules. ESOP expressions are fundamental in the synthesis of quantum circuits, especially in oracles. [15] and [13] describe restricted cases of ESOP expressions which are *Fixed-Polarity Reed-Muller* (FPRM) and *Kronecker Reed-Muller* (KRM) forms. Because finding the minimum ESOP for many variables is very difficult, computer scientists and engineers use algorithms to minimize FPRM and KRM expressions as convenient approximations. Every binary function can be expanded in one of three different ways for every variable to yield new representations of the function. The Shannon expansion, which is the simplest of these three, is given in Equation (2),

$$f(x_1, x_2, \dots, x_n) = xf_x \oplus \bar{x}f_{\bar{x}} \quad (2)$$

where x is some input variable of the function and f_x and $f_{\bar{x}}$ are called the *positive* and *negative cofactors* of variable x . They are equivalent to the function evaluated for $x = 1$ and $x = 0$, respectively. In Equation (3), the Positive Davio (pD) expansion is obtained by substituting $\bar{x} = x \oplus 1$ in the Shannon expansion.

$$f(x_1, x_2, \dots, x_n) = x(f_x \oplus f_{\bar{x}}) \oplus f_{\bar{x}} \quad (3)$$

Likewise, applying the equality $x = \bar{x} \oplus 1$ to the Shannon Expansion produces the Negative Davio (nD) expansion, which is given in Equation (4).

$$f(x_1, x_2, \dots, x_n) = \bar{x}(f_x \oplus f_{\bar{x}}) \oplus f_x \quad (4)$$

Applying the pD expansion to every variable of a function produces the *Positive Polarity Reed-Muller* (PPRM) form, in which every variable is represented with non-complemented (positive) literals. To specify other forms of a function, we use *polarity notation* to denote the type of expansion— pD, nD, or Shannon— on each of the variables. Numbers that denote the polarity of a variable symbolize pD as 0 and nD as 1. Polarity 2 for Shannon is represented only in KRM forms. For example, Polarity $x_1x_2x_3$ denotes a three-variable function, where x_i is the integer representative of the order of the expansions applied on each variable. Regardless of the number of variables, there is only one binary vector of polarities where the polarities of all the variables are positive. A two-variable Boolean function in PPRM can be denoted as Polarity 00, where the 0 in the zeroth index represents the first pD expansion on a variable, and the next 0 represents a pD expansion on the second variable of the Boolean expression resulting from the first expansion.

Next, choosing to apply either pD or nD to each variable yields Fixed Polarity Reed-Muller forms, in which each variable is represented with either complemented or non-complemented literals, but not both. Negative polarity is obtained by applying the nD expansion on a variable. For example, Polarity 110 reveals negative polarity on the first variable, negative polarity on the second variable, and positive polarity on the third. Since each variable can be expanded to either pD or nD in FPRM, there are 2^n possible FPRM polarities for a Boolean function of n variables.

Finally, choosing one of the three expansions to apply to each variable of a function produces a Kronecker Reed-Muller form. Applying a Shannon expansion on a variable results in mixed polarity, typically including both complemented and non-complemented literals. Because there are three different ways to expand each variable, there are 3^n possible KRM expansions for a function of n variables.

Problems of finding the minimum binary FPRM and KRM forms are difficult. Often, a complete search through all the forms is necessary if the exact minimum form is expected. This problem is especially challenging for incompletely specified functions. An incompletely specified function is described as in Equation (1) but the values of minterms m_i can be 0, 1, and either $\hat{0}$ or $\hat{1}$ (which denotes a don't-know value). Don't-knows correspond to incomplete information, formally extending from Boolean functions to Boolean relations. There are no algorithms in existence that would find the exact minimum FPRM, KRM or ESOP solutions to incomplete functions for data of practical size, especially data from ML benchmarks. No problem formulation for hybrid ternary/binary forms is also known. In this paper we formulate the optimal hybrid classifier design problem and solve it using a ternary Grover's Algorithm. Binary Grover's Algorithm is a well-known quantum algorithm especially useful for problems in which only exhaustive search is possible. This is relevant for incomplete functions, because EXORing don't-known values

with known values (cares) in butterfly diagrams [13, 20] creates various constrained combinations that are propagated through the diagram. All these combinations should be investigated by backtracking when an exact minimum cost solution is attempted [13, 15]. Therefore, the new quantum spectral approaches to MV learning and circuit design problems become good research candidates because of the property of quantum parallelism resulting from superposition, particularly in the case that one looks for the exact minimum solution or a small complexity solution. Although our approach is of only theoretical value for the current binary quantum computers with small numbers of qubits, it may become practical when quantum computers with many qubits or especially *qudits* (multivalued quantum units of information) will become available.

Observe also, that minimizing expressions in the EXOR domain [23] is one of the standard methods used in all quantum algorithms to realize binary reversible quantum circuits, specifically to create oracles for Grover's Algorithm [2, 4, 9, 12, 13, 16]. In the future, these methods will be also used to realize ternary and hybrid circuits and oracles. Therefore, interestingly we use here an extended Grover's Algorithm to create oracles for future extended Grover's Algorithms.

2.3 Binary butterfly diagrams and circuits

Spectral transforms are well known in spectral analysis, digital image processing, Machine Learning, and logic synthesis. The best-known transforms are Fourier, Sine, Cosine, Walsh, and Haar. For many of these spectral transforms, there exist *butterfly diagrams*. In this paper, a new type of butterfly diagram is introduced for ternary-input, binary-output functions. Butterfly diagrams are used to find the *spectral coefficients* of a certain KRM or FPRM expansion given the minterms of the function. The spectral coefficients correspond to products of literals of the resulting function after the expansion process. Our goal is to find the expression with the minimum number of products of literals terms.

Figure 1 illustrates the three binary KRM expansions' butterfly kernels. A butterfly kernel represents the calculation of an expansion on a single variable function.

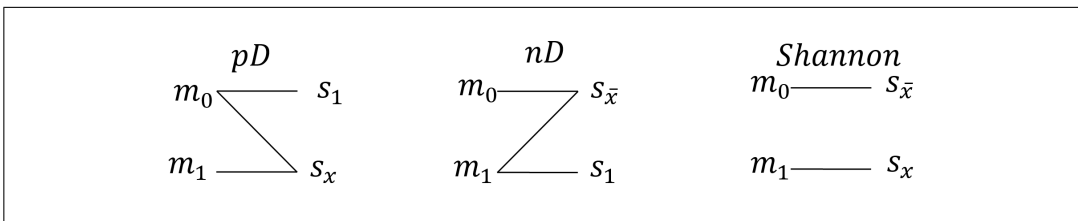


Figure 1: Butterfly kernels of the pD, nD, and Shannon expansions respectively.

m_i , the minterms, and s_i , the spectral coefficients, all hold values of either 1 or 0. To find the set of spectral coefficient values, also referred to as the spectral coefficient values vector from a set of minterm values, horizontal lines in Figure 1 carry the value of the minterm across, while the intersection of two lines indicates an EXOR of the two minterm values on the ends of these lines. The subscripts of s reveal the symbolic representation of spectral coefficients these spectral coefficient values represent, which correspond to some product of literals.

In this paper, we refer to the set of minterm values as simply the *set of minterms*, however, we distinguish the spectral coefficients into two parts: their values are contained in a *spectral coefficient values vector*, and their symbolic representations are referred to in their *symbolic names of spectral coefficients vector* (product of literals).

Parallels can be drawn between a binary expansion's canonical expansion formula and butterfly diagram [15]. For example, in Equation (3) x acts as the coefficient of $f_x \oplus f_{\bar{x}}$ while 1 is the coefficient for the other literal $f_{\bar{x}}$. The expanded Boolean function is achieved from an EXOR on the resulting terms after multiplying each spectral coefficient value with its representative product of literals. Notice that when the spectral coefficient value is 0, the term is omitted from the resulting equation. These similarities between butterfly kernels and their respective expansion formulas are present in the nD and Shannon expansions as well. In highlighting this correspondence, it becomes clear how we can derive the ternary-input binary-output butterfly kernel— and therefore reversible circuit— given the expansion's complete equation and vice versa.

Kernels only act on a function of a single variable. For functions with more variables, the kernels can be consecutively implemented, called butterfly diagrams [13, 15, 20]. Variable columns, each corresponding to a transformation on a certain variable of the function, are arranged in reverse order to how they are applied on the original function. Therefore, x_1 is transformed in the second column of the diagram, x_2 in the first.

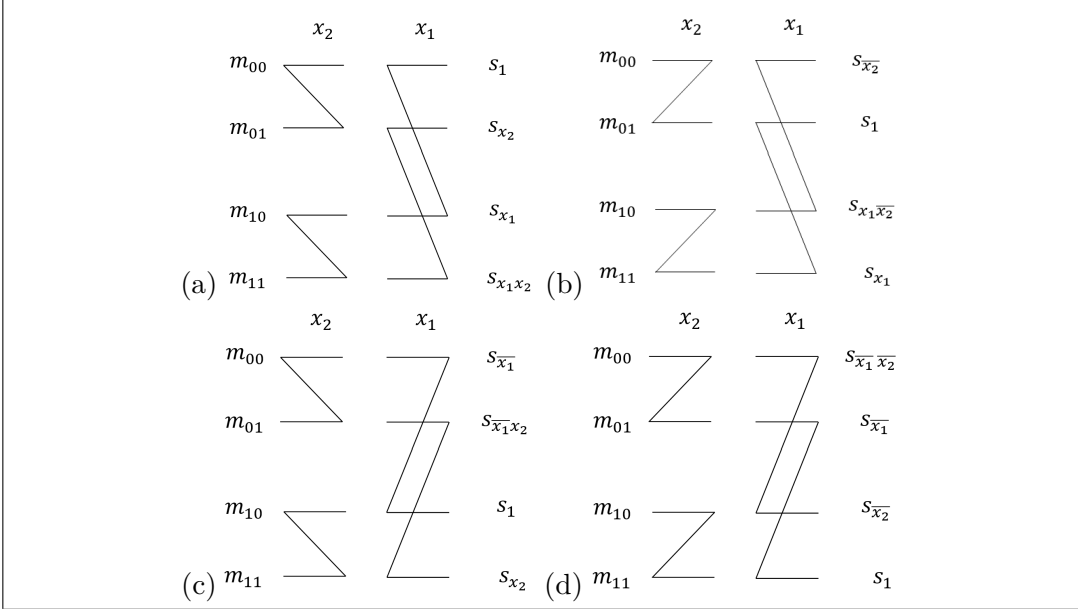


Figure 2: All FPRM forms of a two-variable binary function. (a) Polarity 00; (b) Polarity 01; (c) Polarity 10; (d) Polarity 11.

In Figure 2, there are four minterms for two variables, where one kernel connects two minterms, so two kernels are present in each column. Assume the distance between each minterm as a single unit. In the first column, the first kernel is stretched 2^0 unit, from m_0 to m_1 . In the second column, the first kernel is stretched from m_0 to m_2 , or 2^1 units. The stretching distance increases by 2^k with each column, where k is the number of columns away from the unstretched kernels.

After transforming the set of minterms to its corresponding set of spectral coefficient values, the product of literals corresponding to each coefficient value in the formula must be determined to construct the resulting Boolean expression. Determining product literals for multiple variables is more challenging than simply referring to the expansion formula as done with a single variable. However, the same concepts can still be used. Notice how the subscripts of s in Figure 1, the set of coefficients can be written as *polarity(coefficients) vectors*: $[1, x]$ for pD, $[\bar{x}, 1]$ for nD, and $[\bar{x}, x]$ for Shannon. Applying Tensor (Kronecker) products to the polarity vectors results in a symbolic names of spectral coefficients vector that encompass all the variables. To find the symbolic names of spectral coefficients vector for Polarity 00 of Figure 2: $[1, x_1] \otimes [1, x_2] = [1 \cdot 1, 1 \cdot x_2, x_1 \cdot 1, x_1 \cdot x_2] = [1, x_2, x_1, x_1x_2]$.

Choosing to use either the pD, nD, or Shannon kernels in each column yields different expansions on each variable. As an example, let us evaluate the function

in Equation (5)

$$F(a, b, c) = abc \oplus \bar{c} \quad (5)$$

on Polarity 012—first algebraically, then using butterfly diagrams. From its polarity notation, we can recognize pD will be used to expand the variable a , then nD on variable b , and finally Shannon expansion on variable c .

To algebraically expand the Boolean function, we first expand pD on variable a . Instead of the variable x in Equation (3), the positive and negative cofactors will be derived in respect to a . This can be calculated by substituting $a = 1$ and $a = 0$ into Equation (5). After simplifying with some ESOP rules, $F_a = bc \oplus \bar{c}$ and $F_{\bar{a}} = \bar{c}$. Thus, we can assemble $F_1 = a((bc \oplus \bar{c}) \oplus \bar{c}) \oplus \bar{c}$, which is Equation (5) after a pD expansion on variable a . Here, we refer to Equation (4): x now refers to the variable b . Undergoing a similar process as to the previous expansion, the positive cofactor is calculated by substituting $b = 1$ into F_1 , while the negative cofactor is calculated by substituting $b = 0$ into it instead. This results in $F_b = ac \oplus \bar{c}$ and $F_{\bar{b}} = \bar{c}$, respectively. Therefore, Equation (5) after having undergone a pD expansion on variable a and nD on variable b is $F_2 = \bar{b}((ac \oplus \bar{c}) \oplus \bar{c}) \oplus (ac \oplus \bar{c})$. Finally, mixed polarity is applied to variable c through Shannon. In Equation (2), x now refers to the variable c . After substituting $c = 1$ and $c = 0$ into F_2 to find the cofactors of c , we find that $F_c = a\bar{b} \oplus a$ and $F_{\bar{c}} = 1$. These literals in the Shannon formula become $F_3 = c(a\bar{b} \oplus a) \oplus \bar{c}(1)$, simplified to

$$F_3 = a\bar{b}c \oplus ac \oplus \bar{c} \quad (6)$$

as the resulting function from expanding Equation (5) on Polarity 012.

This same process can be replicated through evaluating a butterfly diagram. The minterms (inputs to the butterfly diagram) of Equation (5) are shown in Figure 3.

		c	0	1
ab				
00			1	0
01			1	0
11			1	1
10			1	0

Figure 3: Karnaugh Map of $F(a, b, c) = abc \oplus \bar{c}$.

The three columns of the butterfly diagram represent the three variables. Since the minterms are transformed in reverse from which variables are expanded algebraically, the first column consists of four unstretched butterfly kernels representative of a Shannon expansion. The second column consists of four stretched nD

kernels. The third column consists of pD kernels stretched even further. The setup is as illustrated in Figure 4.

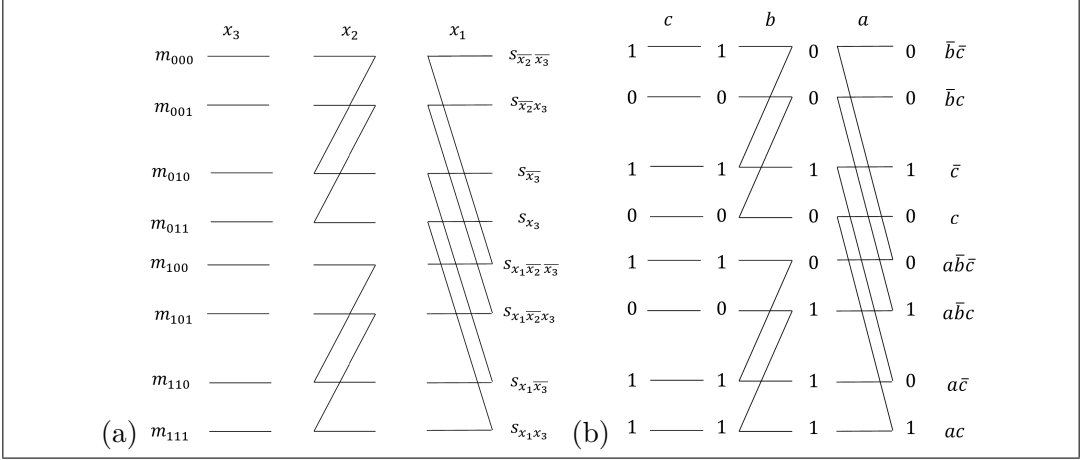


Figure 4: (a) Butterfly diagram of Polarity 012 on an arbitrary three-variable function. (b) Evaluated butterfly diagram of Polarity 012 on $F(a, b, c) = abc \oplus \bar{c}$

Values of minterms in natural order are at the left of the butterfly diagram and the values and symbols of spectral coefficients at the right. The symbolic names of spectral coefficients vector in Figure 4 is calculated in Equation (7).

$$[1, a] \otimes [\bar{b}, 1] \otimes [\bar{c}, c] = [\bar{b}\bar{c}, \bar{b}c, \bar{c}, c, a\bar{b}\bar{c}, a\bar{b}c, a\bar{c}, ac] \quad (7)$$

Then, we multiply the vector of spectral coefficient values with its respective symbolic names of spectral coefficients vector.

$$0 \cdot \bar{b}\bar{c} \oplus 0 \cdot \bar{b}c \oplus 1 \cdot \bar{c} \oplus 0 \cdot c \oplus 0 \cdot a\bar{b}\bar{c} \oplus 1 \cdot a\bar{b}c \oplus 0 \cdot a\bar{c} \oplus 1 \cdot ac \quad (8)$$

simplify, and we obtain:

$$F(a, b, c) = \bar{c} \oplus a\bar{b}c \oplus ac \quad (9)$$

as the final function of transforming the minterms of Equation (5) on Polarity 012 using a butterfly diagram. When comparing the resulting expressions from the two methods, it becomes clear they are equivalent. Additionally, realize that the expression from Equation (9) is more expensive than Equation (5).

Most authors realize butterflies in software, however, [15] and [13] realize butterflies in reversible circuits that are used inside the Grover's Algorithm oracle. This raises the idea that any type of butterfly diagram can be incorporated into a quantum oracle that can be used inside a quantum algorithm. This algorithm can be

the Grover's Algorithm or a Grover's Algorithm generalized to MV logic. It can be also a Quantum Walk Algorithm [1] or another quantum search algorithm such as many existing generalizations of Grover's Algorithm. Following these two references, we also realize butterflies as reversible circuits. For example, a single kernel of pD realized as a reversible circuit is shown in Figure 5.

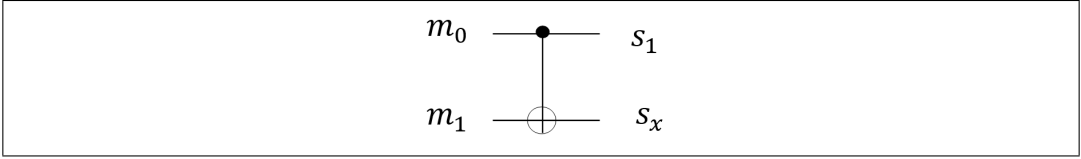


Figure 5: A single pD kernel realized as a reversible circuit.

This is just for pD, but if we want to select either pD or nD, then we will need to do as is done in Figure 6: when the polarity line p_a is equal to 0, nD is chosen, when it is one, pD is chosen.

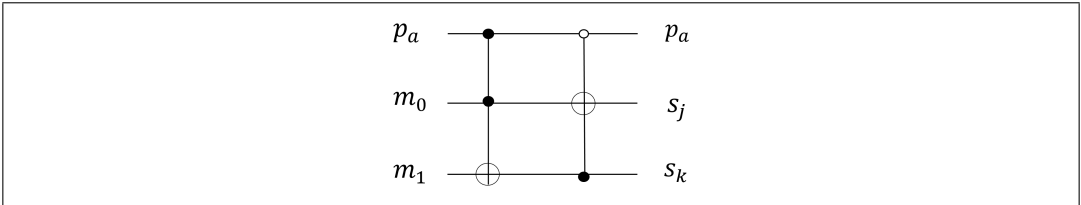


Figure 6: Both expansions are realized in this circuit, and one or the other can be chosen using the control line, p_a .

This circuit realizes all FPRM expansions of a single variable function. If the polarity line is ternary, the control value of 2 represents the Shannon expansion, and all KRM expansions can be realized. Butterfly diagrams for functions of more than one variable can be realized by repeating and stretching this circuit. For every variable, a qubit will need to be added for control of polarity selection. These controlled variable columns are called *partial kernels*. Figure 7 shows a circuit much like that in Figure 6, but for a function of two variables. It also utilizes ternary polarity lines so that 0 may represent the pD expansion, 1 the nD expansion, and 2 the Shannon expansion.

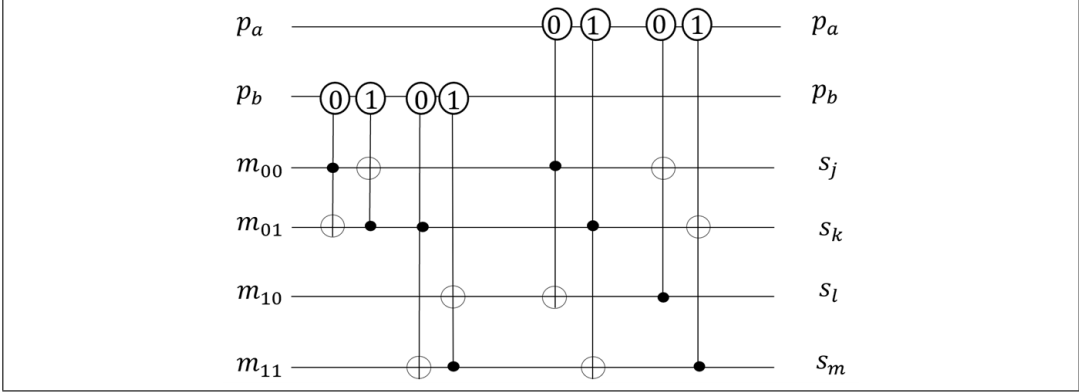


Figure 7: This circuit realizes either pD, nD, or Shannon expansions on each of two variables of the function specified by the binary minterms (m_{jk}). The polarity lines p_a and p_b are ternary, therefore, a different notation for these control qubits is used to distinguish ternary from binary qubits used in the binary minterms.

The first column of the Kronecker Reed-Muller expansion forms are controlled by p_b and transforms variable b , while the second set, controlled by p_a does so for a . Because no transformation is applied during the Shannon expansion, no gates are activated when both polarity lines have a value of 2. The two ternary polarity lines are used to choose one of the 3^2 possible KRM expansions. The outputs of this circuit are the polarity vector $[p_a, p_b]$ and the vector of spectral coefficient values $[s_j, s_k, s_l, s_m]$. Figure 7 can be expanded to an arbitrary number of variables, using the butterfly diagrams to determine the structure.

3 Algorithm for ternary-input KRM expression minimization

In past designs of KRM minimization and in our design, Grover's Algorithm has a quantum oracle that uses the so-called "Kronecker Reed-Muller Processor" or *KRM Processor* [13, 15]. The oracle used for finding the minimal expansion on a given ternary-input binary-output function is the core of our design and is described in this section. All other aspects of the algorithm are standard and will be reviewed, as well as how the algorithm works with our oracle, in Section 4.

3.1 Ternary-input KRM Processor

In here and in Section 3.1.1, we will first go over the ternary generalizations of the Shannon and Davio expansions. Then, in 3.1.2 we will look at the corresponding butterflies, individual circuits, and how these can be used to create the ternary-input, binary-output KRM Processor.

The Shannon expansion for a ternary-input, binary-output function with respect to variable x_i can be written as in Equation (10):

$$f(x_1, x_2, \dots, x_n) = x^0 f_{x^0} \oplus x^1 f_{x^1} \oplus x^2 f_{x^2} \quad (10)$$

This is a generalization of the binary Shannon expansion: all the function's (disjoint) cofactors times corresponding literals combined by EXORs. Some papers use the matrix representation of these equations to discuss the KRM expansions, which our ternary-input expansions can be written in as well. This notation is most well-known in matrix programming language MATLAB [11], but is also used in numerical computation language GNU Octave [7]. The Karnaugh Map matrix of Equation 10 is:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (11)$$

where the argument variables are ternary and the function values (as shown) are binary. Notice that these matrices communicates the same information as our aforementioned polarity/coefficient vectors, so the discussion of matrices to our ternary expansions will only be covered briefly in this paper.

In ternary logic, we have constant 1 and six literals.

x	x^0	x^1	x^2	x^{01}	x^{02}	x^{12}	1
0	1	0	0	1	1	0	1
1	0	1	0	1	0	1	1
2	0	0	1	0	1	1	1

Figure 8: Table of all vectors of ternary literals.

In Figure 8, the ternary literals are depicted as x^0 , x^1 , x^2 , x^{01} , x^{02} , x^{12} and 1. This contrasts binary logic, which has two polarities— x and \bar{x} . Equation (12) denotes examples of how these literals can be substituted for each other.

$$\begin{aligned}
 x^0 \oplus x^1 &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = x^{01} \\
 x^{12} &= x^1 \oplus x^2 = 1 \oplus x^0
 \end{aligned} \tag{12}$$

3.1.1 How many expansions do we need for ternary-input binary-output logic?

To create the oracle for Hybrid KRM, we need to know all equivalents of the Davio and Shannon expansions. Because of the increased number of polarities, there will be 27 polarities of ternary-input, binary-output generalized Davio expansions plus one Ternary Shannon Expansion. We can represent each literal as a binary vector. Finding all possible polarities of expansions becomes a problem of finding sets of three vectors for which corresponding indices can be combined with EXORs to yield those used in the Ternary Shannon Expansion. Furthermore, these sets must be *linearly independent*. A set is linearly independent if no vector in the set can be obtained by a linear combination of other vectors. Therefore, in the case of our 28 expansions, all sets of three linearly dependent vectors, those in which one can be reached by a linear combination of two others, will only contain ones in two positions. This leaves no way to obtain a 1 by EXOR in the final position. All linearly independent sets of vectors will have at least a single 1 in every position. If every vector contains a 1 in the same position, a 0 in that position can always be reached by the EXOR of two 1s. Therefore, by finding every linearly independent set of vectors, we were able to create every possible out of 27 ternary Davio expansions. We now derive these expansions.

Theorem. In ternary-input, binary-output logic there are 28 expansions (including the original Ternary Shannon).

Proof. We begin by dividing the vectors for every ternary-input variable into

three types:

$$1|A^0, A^1, A^2|A^{01}, A^{02}, A^{12} \quad (13)$$

A will be used to indicate a generalization to all variables. This A refers to the variable name x if there is one variable, but A can also be a , b , x_2 , and y in this paper. The first is the 1 vector. We will call the second and third types of vectors Type 1 and Type 2 vectors. We start counting sets of vectors with those that include the 1 vector. From here, we first count those with only 1 and Type 1 vectors, and those with only 1 and Type 2 vectors, as none of these sets will be linearly dependent. We get Equation (14).

$$[1, A^1, A^2], [1, A^0, A^2], [1, A^0, A^1] \quad (14)$$

Next, Equation (15).

$$[1, A^{01}, A^{02}], [1, A^{01}, A^{12}], [1, A^{02}, A^{12}] \quad (15)$$

Now we look at those with 1, one Type 1 vector, and one Type 2 vector. Simply listing all the vectors that fit this description will give some that are linearly dependent, so some must be omitted. First, we list each combination of 1 and a Type 2 vector. For every Type 1 and Type 2 vector, there exists one Type 1 vector that cannot be the third member of the set because it will create a linearly dependent set. Equation (16) is a total of six transformation forms.

$$[A^0, A^{01}, 1], [A^{01}, A^1, 1], [A^0, 1, A^{02}], [A^{02}, 1, A^2], [1, A^1, A^{12}], [1, A^{12}, A^2] \quad (16)$$

Now we count those with two Type 1 vectors and one Type 2 vector. This is done in Equation (17) in a similar way to the previous group. We get six transforms.

$$[A^0, A^1, A^{02}], [A^0, A^1, A^{12}], [A^{01}, A^1, A^2], [A^{02}, A^1, A^2], [A^0, A^{01}, A^2], [A^0, A^{12}, A^2] \quad (17)$$

Then we look at sets of two Type 2 vectors and one Type 1. No matter which pair of Type 2 vectors we chose, we can add any of the three Type 1 vectors to it. We get 9 transformations in Equation (18).

$$[A^{01}, A^{02}, A^0 \text{ or } A^1 \text{ or } A^2], [A^{01}, A^{02}, A^0 \text{ or } A^1 \text{ or } A^2], [A^{01}, A^{12}, A^0 \text{ or } A^1 \text{ or } A^2] \quad (18)$$

The final two sets in Equation (19) are simple.

$$[A^0, A^1, A^2], [A^{01}, A^{02}, A^{12}] \quad (19)$$

Counting the vectors, we get total combinations = $3 + 3 + 6 + 6 + 9 + 2 = 29$.

Because the matrix must be non-singular, the transformation $[A^{01}, A^{02}, A^{12}]$ is excluded. Thus, we have a total of 28 expansions for Ternary-Input, Binary-Output Kronecker Polarity Reed-Muller Expansions.

The reader is advised to compare Equation (14) with Equation (19) above with coefficient vectors, partial butterfly kernels, and circuits in Table 12, which are multiplexed to create a combined single variable oracle kernel (Figure 14(a)). For convenience, we will now refer to these vectors of literals as the result of polarities, or coefficient vectors, by their equation number as in the order of how they were realized above (these names are restated in Table 12). Equation (14)a and Equation (19)b are examples of coefficient vectors.

Other proof method. Because the columns of such matrices must be linearly independent, the matrices must be invertible. Furthermore, the order of the rows in the matrices does not matter. Changing this order does not alter the expansion that the matrix performs, only changes the order of its outputs. The number of generalized expansions, including Shannon, is the number of invertible binary matrices of size 3×3 divided by the number of possible arrangements of rows of a 3×3 matrix. Given that the number of invertible, binary, square matrices of a given size can be found using the equation: $\prod_{k=0}^{n-1} (2^n - 2^k)$. Where n is the size of the matrix, the number of generalized expansions can be written: $\frac{\prod_{k=0}^{3-1} (2^3 - 2^k)}{6} = 28$. This is the base of our expansions. A good base means every function realized in one unique way. Thus, for a function of n variables there are 27^n ternary-input FPRMs and 28^n ternary-input KRMs. They are constructed analogously as the binary case from [15] and [13], using respective butterfly diagrams, decision diagrams or other representation methods.

3.1.2 The procedure to find the minimal quantum circuit for every equation.

First, let us discuss the expansions for a completely specified function of a single ternary variable. Once we have formulated the sets of literals, the expansion can be written by substituting the A^0 , A^1 , and A^2 literals of the Ternary Shannon Expansion with EXORs of other literals.

For example, using the set $[A^0, A^1, 1]$, which we refer to in this paper as Equation (14)c, we can write the expansion as $f(x_1, x_2, \dots, x_n) = x^0 f_{x^0} \oplus x^1 f_{x^1} \oplus (x^0 \oplus x^1 \oplus 1) f_{x^2}$ simplified to $f(x_1, x_2, \dots, x_n) = x^0 (f_{x^0} \oplus f_{x^2}) \oplus x^1 (f_{x^1} \oplus f_{x^2}) \oplus f_{x^2}$ where f_{x^0} , f_{x^1} and f_{x^2} are cofactors of f with respect to selected variable x_i . Their binary values are denoted as m_0, m_1, m_2 in Table 12 and x_0, x_1, x_2 are literals (of the selected variable).

We can find realizations of all the kernels for a single variable in the same way as in [15] and [13]. Therefore, the ternary butterfly acts the same way, only its size and shape are different than in the binary case. Remembering that the combination of cofactors computes the spectral coefficients, the previous expansions Equation (14) and Equation (19) can be used to create all of the 27 single-variable ternary FPRM butterfly kernels. Figure 9 illustrates the butterfly kernel and reversible circuit for Equation (14)c. Referring to Figure 9, Figure 11, and Figure 12, m_i and f_{x^i} are equivalent for a one variable butterfly kernel, they are just in different notation.

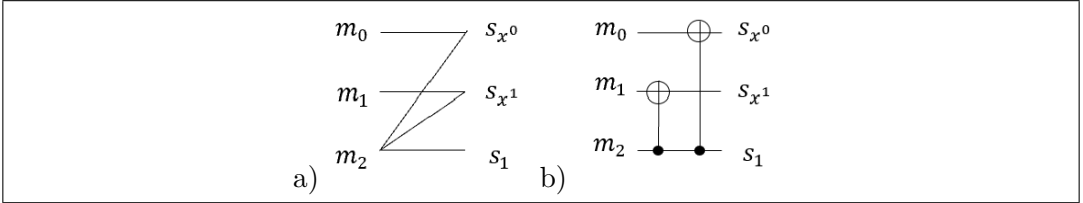


Figure 9: a) The butterfly kernel of Equation (14)c. b) The reversible circuit of Equation (14)c.

Following the pattern of the binary expansions, the EXOR of cofactors in the equation produce an intersection point in the butterfly kernel. This intersection translates to a controlled-NOT gate in the reversible circuit.

Polarity Vector: $[1, A^{02}, A^{01}] \rightarrow [1, x^{02}, x^{01}]$

Ternary Shannon $[A^0, A^1, A^2]$:

$$f(x_1, x_2, \dots, x_n) = x^0 f_{x^0} \oplus x^1 f_{x^1} \oplus x^2 f_{x^2}$$

Substitutions:

$$1 \oplus x^{01} \oplus x^{02} = x^0 \oplus x^1 \oplus x^2 \oplus (x^0 \oplus x^1) \oplus (x^0 \oplus x^2) = x^0$$

$$1 \oplus x^{02} = x^0 \oplus x^1 \oplus x^2 \oplus (x^0 \oplus x^2) = x^1$$

$$1 \oplus x^{01} = x^0 \oplus x^1 \oplus x^2 \oplus (x^0 \oplus x^1) = x^2$$

Substitute into Ternary Shannon:

$$f(x_1, x_2, \dots, x_n) = (1 \oplus x^{01} \oplus x^{02}) f_{x^0} \oplus (1 \oplus x^{02}) f_{x^1} \oplus (1 \oplus x^{01}) f_{x^2}$$

Complete Ternary Expansion Equation for $[1, A^{02}, A^{01}]$:

$$f(x_1, x_2, \dots, x_n) = (f_{x^0} \oplus f_{x^1} \oplus f_{x^2}) \oplus x^{02} (f_{x^0} \oplus f_{x^1}) \oplus x^{01} (f_{x^0} \oplus f_{x^2})$$

Figure 10: A more detailed and systematic approach to realizing the complete ternary expansion equation for Equation (15)a, or $[1, A^{01}, A^{02}]$, from Ternary Shannon.

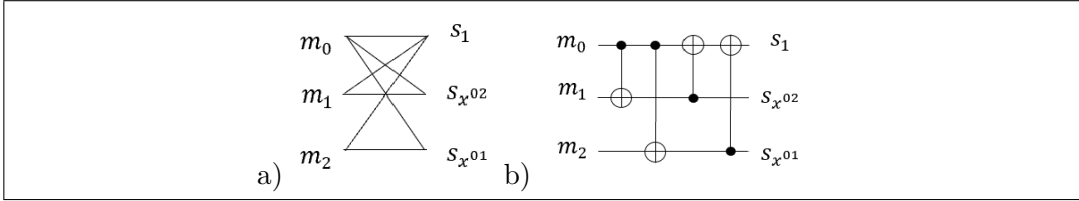
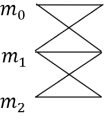
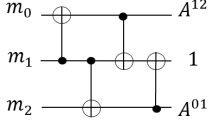
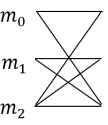
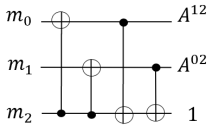
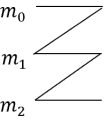
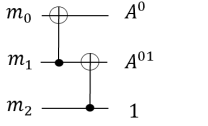
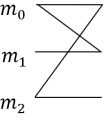
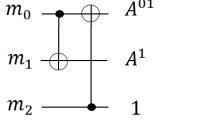
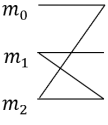
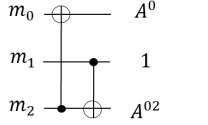


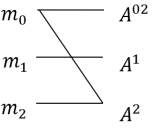
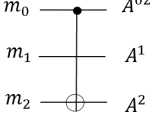
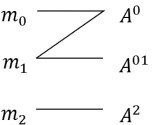
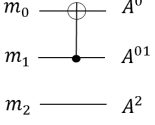
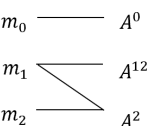
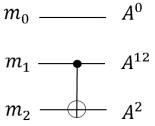
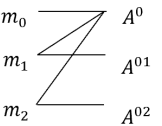
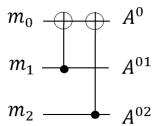
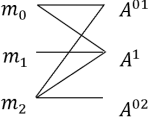
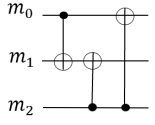
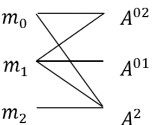
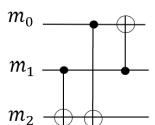
Figure 11: a) The butterfly kernel of Equation (15)a. b) The reversible circuit of Equation (15)a.

The 3 expansions, defined by the groups of literals, $[1, A^{01}, A^{02}]$, $[1, A^{01}, A^{12}]$ and $[1, A^{12}, A^{02}]$, have more complicated reversible circuits because they utilize more Toffoli gates. However, we have developed this systematic approach that can still as simply realize equation, butterfly kernel, and reversible circuit. In Figure 12, we realize all 29 linearly independent polarity vectors to their corresponding expansion equations, butterfly kernels, and reversible circuits in this way.

Equation Number	Complete Ternary Expansion Equation with Respect to Variable x_i	Polarity Vector	Butterfly Kernel	Reversible Circuit
(19) a	$f(x_1, x_2, \dots, x_n)$ $= x^0 f_{x^0} \oplus x^1 f_{x^1} \oplus x^2 f_{x^2}$	$[A^0, A^1, A^2]$	m_0 ——— A^0 m_1 ——— A^1 m_2 ——— A^2	m_0 ——— A^0 m_1 ——— A^1 m_2 ——— A^2
(14) a	$f(x_1, x_2, \dots, x_n)$ $= f_{x^0} \oplus x^1 (f_{x^0} \oplus f_{x^1}) \oplus x^2 (f_{x^0} \oplus f_{x^2})$	$[1, A^1, A^2]$	m_0 ——— 1 m_1 ——— A^1 m_2 ——— A^2	m_0 ——— 1 m_1 ——— A^1 m_2 ——— A^2
(14) b	$f(x_1, x_2, \dots, x_n)$ $= x^0 (f_{x^0} \oplus f_{x^1}) \oplus f_{x^1} \oplus x^2 (f_{x^1} \oplus f_{x^2})$	$[A^0, 1, A^2]$	m_0 ——— A^0 m_1 ——— 1 m_2 ——— A^2	m_0 ——— A^0 m_1 ——— 1 m_2 ——— A^2
(14) c	$f(x_1, x_2, \dots, x_n)$ $= x^0 (f_{x^0} \oplus f_{x^2}) \oplus x^1 (f_{x^1} \oplus f_{x^2}) \oplus f_{x^2}$	$[A^0, A^1, 1]$	m_0 ——— A^0 m_1 ——— A^1 m_2 ——— 1	m_0 ——— A^0 m_1 ——— A^1 m_2 ——— 1
(15) a	$f(x_1, x_2, \dots, x_n)$ $= (f_{x^0} \oplus f_{x^1} \oplus f_{x^2}) \oplus x^{02} (f_{x^0} \oplus f_{x^1}) \oplus x^{01} (f_{x^0} \oplus f_{x^2})$	$[1, A^{02}, A^{01}]$	m_0 ——— 1 m_1 ——— A^{02} m_2 ——— A^{01}	m_0 ——— 1 m_1 ——— A^{02} m_2 ——— A^{01}

(15) b	$f(x_1, x_2, \dots, x_n)$ $= x^{12}(f_{x^0} \oplus f_{x^1}) \oplus (f_{x^0} \oplus f_{x^1} \oplus f_{x^2})$ $\oplus x^{01}(f_{x^1} \oplus f_{x^2})$	$[A^{12}, 1, A^{01}]$	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^{12}</div> <div>m_1 1</div> <div>m_2 A^{01}</div> </div>	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^{12}</div> <div>m_1 1</div> <div>m_2 A^{01}</div> </div>
(15) c	$f(x_1, x_2, \dots, x_n)$ $= x^{12}(f_{x^0} \oplus f_{x^2}) \oplus x^{02}(f_{x^1} \oplus f_{x^2}) \oplus$ $(f_{x^0} \oplus f_{x^1} \oplus f_{x^2})$	$[A^{12}, A^{02}, 1]$	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^{12}</div> <div>m_1 A^{02}</div> <div>m_2 1</div> </div>	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^{12}</div> <div>m_1 A^{02}</div> <div>m_2 1</div> </div>
(16) a	$f(x_1, x_2, \dots, x_n)$ $= x^0(f_{x^0} \oplus f_{x^1}) \oplus x^{01}(f_{x^1} \oplus f_{x^2}) \oplus f_{x^2}$	$[A^0, A^{01}, 1]$	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^0</div> <div>m_1 A^{01}</div> <div>m_2 1</div> </div>	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^0</div> <div>m_1 A^{01}</div> <div>m_2 1</div> </div>
(16) b	$f(x_1, x_2, \dots, x_n)$ $= x^{01}(f_{x^0} \oplus f_{x^2}) \oplus x^1(f_{x^0} \oplus f_{x^1}) \oplus f_{x^2}$	$[A^{01}, A^1, 1]$	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^{01}</div> <div>m_1 A^1</div> <div>m_2 1</div> </div>	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^{01}</div> <div>m_1 A^1</div> <div>m_2 1</div> </div>
(16) c	$f(x_1, x_2, \dots, x_n)$ $= x^0(f_{x^0} \oplus f_{x^2}) \oplus f_{x^1} \oplus x^{02}(f_{x^1} \oplus f_{x^2})$	$[A^0, 1, A^{02}]$	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^0</div> <div>m_1 1</div> <div>m_2 A^{02}</div> </div>	 <div style="display: flex; flex-direction: column; align-items: flex-end;"> <div>m_0 A^0</div> <div>m_1 1</div> <div>m_2 A^{02}</div> </div>

(16) d	$f(x_1, x_2, \dots x_n)$ $= x^{02}(f_{x^0} \oplus f_{x^1}) \oplus f_{x^1} \oplus x^2(f_{x^0} \oplus f_{x^2})$	$[A^{02}, 1, A^2]$		
(16) e	$f(x_1, x_2, \dots x_n)$ $= f_{x^0} \oplus x^1(f_{x^1} \oplus f_{x^2}) \oplus x^{12}(f_{x^0} \oplus f_{x^2})$	$[1, A^1, A^{12}]$		
(16)f	$f(x_1, x_2, \dots x_n)$ $= f_{x^0} \oplus x^{12}(f_{x^0} \oplus f_{x^1}) \oplus x^2(f_{x^1} \oplus f_{x^2})$	$[1, A^{12}, A^2]$		
(17) a	$f(x_1, x_2, \dots x_n)$ $= x^0(f_{x^0} \oplus f_{x^2}) \oplus x^1 f_{x^1} \oplus x^{02} f_{x^2}$	$[A^0, A^1, A^{02}]$		
(17) b	$f(x_1, x_2, \dots x_n)$ $= x^0 f_{x^0} \oplus x^1(f_{x^1} \oplus f_{x^2}) \oplus x^{12} f_{x^2}$	$[A^0, A^1, A^{12}]$		
(17) c	$f(x_1, x_2, \dots x_n)$ $= x^{01} f_{x^0} \oplus x^1(f_{x^0} \oplus f_{x^1}) \oplus x^2 f_{x^2}$	$[A^{01}, A^1, A^2]$		

(17) d	$f(x_1, x_2, \dots, x_n)$ $= x^{02} f_{x^0} \oplus x^1 f_{x^1} \oplus x^2 (f_{x^0} \oplus f_{x^2})$	$[A^{02}, A^1, A^2]$		
(17) e	$f(x_1, x_2, \dots, x_n)$ $= x^0 (f_{x^0} \oplus f_{x^1}) \oplus x^{01} f_{x^1} \oplus x^2 f_{x^2}$	$[A^0, A^{01}, A^2]$		
(17) f	$f(x_1, x_2, \dots, x_n)$ $= x^0 f_{x^0} \oplus x^{12} f_{x^1} \oplus x^2 (f_{x^1} \oplus f_{x^2})$	$[A^0, A^{12}, A^2]$		
(18) a	$f(x_1, x_2, \dots, x_n)$ $= x^1 (f_{x^0} \oplus f_{x^1} \oplus f_{x^2}) \oplus x^{01} f_{x^1} \oplus x^{02} f_{x^2}$	$[A^0, A^{01}, A^{02}]$		
(18) b	$f(x_1, x_2, \dots, x_n)$ $= x^{01} (f_{x^0} \oplus f_{x^2}) \oplus x^1 (f_{x^0} \oplus f_{x^1} \oplus f_{x^2}) \oplus x^{02} f_{x^2}$	$[A^{01}, A^1, A^{02}]$		
(18) c	$f(x_1, x_2, \dots, x_n)$ $= x^{02} (f_{x^0} \oplus f_{x^1}) \oplus x^{01} f_{x^1} \oplus x^2 (f_{x^0} \oplus f_{x^1} \oplus f_{x^2})$	$[A^{02}, A^{01}, A^2]$		

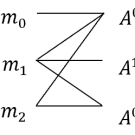
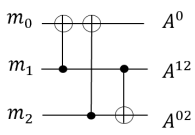
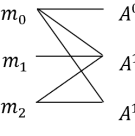
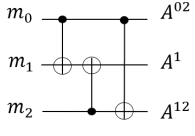
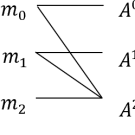
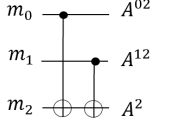
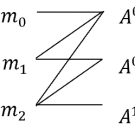
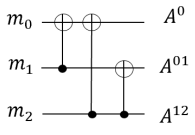
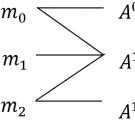
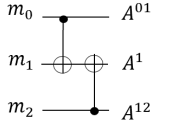
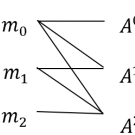
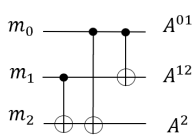
(18) d	$f(x_1, x_2, \dots, x_n)$ $= x^0(f_{x^0} \oplus f_{x^1} \oplus f_{x^2}) \oplus x^{12} f_{x^1}$ $\oplus x^{02}(f_{x^1} \oplus f_{x^2})$	$[A^0, A^{12}, A^{02}]$		
(18) e	$f(x_1, x_2, \dots, x_n)$ $= x^{02} f_{x^0} \oplus x^1(f_{x^0} \oplus f_{x^1} \oplus f_{x^2})$ $\oplus x^{12}(f_{x^0} \oplus f_{x^2})$	$[A^{02}, A^1, A^{12}]$		
(18) f	$f(x_1, x_2, \dots, x_n)$ $= x^{02} f_{x^0} \oplus x^{12} f_{x^1} \oplus x^2$ $(f_{x^0} \oplus f_{x^1} \oplus f_{x^2})$	$[A^{02}, A^{12}, A^2]$		
(18) g	$f(x_1, x_2, \dots, x_n)$ $= x^0(f_{x^0} \oplus f_{x^1} \oplus f_{x^2}) \oplus x^{01}$ $(f_{x^1} \oplus f_{x^2}) \oplus x^{12} f_{x^2}$	$[A^0, A^{01}, A^{12}]$		
(18) h	$f(x_1, x_2, \dots, x_n)$ $= x^{01} f_{x^0} \oplus x^1(f_{x^0} \oplus f_{x^1} \oplus f_{x^2}) \oplus x^{12}$ f_{x^2}	$[A^{01}, A^1, A^{12}]$		
(18) i	$f(x_1, x_2, \dots, x_n)$ $= x^{01} f_{x^0} \oplus x^{12}(f_{x^0} \oplus f_{x^1}) \oplus x^2$ $(f_{x^0} \oplus f_{x^1} \oplus f_{x^2})$	$[A^{01}, A^{12}, A^2]$		

Table 12: All ternary expansion equations and their respective coefficient polarity vectors, butterfly kernels, and reversible circuits for partial kernels.

As with binary, we can realize the butterflies as binary circuits that operate on literals which convert from ternary to binary. The butterflies for all 28 KRM expansions are shown in Table 12. These are ternary counterparts of binary butterflies and kernels from [13]. The minterms of the Karnaugh Map, m_0, m_1, m_2 , are binary values for a ternary-input, binary-output function, as they completely characterize the function. The polarity coefficients are placed within the butterfly kernel and reversible circuit next to their corresponding cofactors as they are in the expansion equation. In [13], the partial kernel circuits for the binary Positive Davio, Negative Davio and Shannon expansions were combined to a single quantum kernel circuit. Here we do the same, but we have to combine 27 partial kernels for FPRM-like expansions and 28 partial kernels for KRM-like ternary expansions. We combine all single-expansions partial kernels to a single one-variable kernel, in which the top control lines control the polarity expansion of each variable. This is called a *quantum multiplexer*.

In the binary FPRM circuit we need just one qubit to control one of two expansions. In KRM we need a ternary control because we have three expansions. However, now with 27 or 28 expansions the design of the controller becomes complicated. It can be formulated as a new type of encoding problem to minimize the complete one-variable kernel. This problem is a separate research topic, so in this paper we select simplified solutions based on ternary/binary controls and simple encoding based on quantum multiplexers [20]. A ternary reversible multiplexer is discussed in [1].

Realizing the ternary FPRM kernel is still relatively easy. $3^3 = 27$, thus we need three ternary qudits (called *qutrits*) to select the expansion type and its corresponding partial kernel linear circuit. As the controls are ternary the Grover's Algorithm uses *Chrestenson's gates* instead standard Hadamard for these variables [13].

When one wants to find the exact minimum KRM the design becomes more complicated, as there are 28 possible expansions, including Ternary Shannon. The control line for every variable can be hypothetically a 28-valued qudit, but this is unrealistic in current quantum technologies. Thus, control of each combined target kernel for a variable is constructed here from 5 binary control lines ($2^5 = 32 > 28$). In the presented design we use a reversible multiplexer that uses 28 out of 32 combinations, but this circuit can be optimized further in several ways, which is not a topic for this paper.

3.2 Examples of oracles

Here, we create oracles with our ternary-input binary-output butterfly kernels following the structure of their complete butterfly diagrams as was described previously

in binary logic. As a simple example, a butterfly diagram for two ternary-input variables a, b is shown in Figure 13. This butterfly has coefficient (polarity) vector $[1, A^1, A^2]$ for variable a and coefficient vector $[A^0, 1, A^2]$ for variable b . This size is the most of what we can simulate right now because of the limited power of quantum simulators.

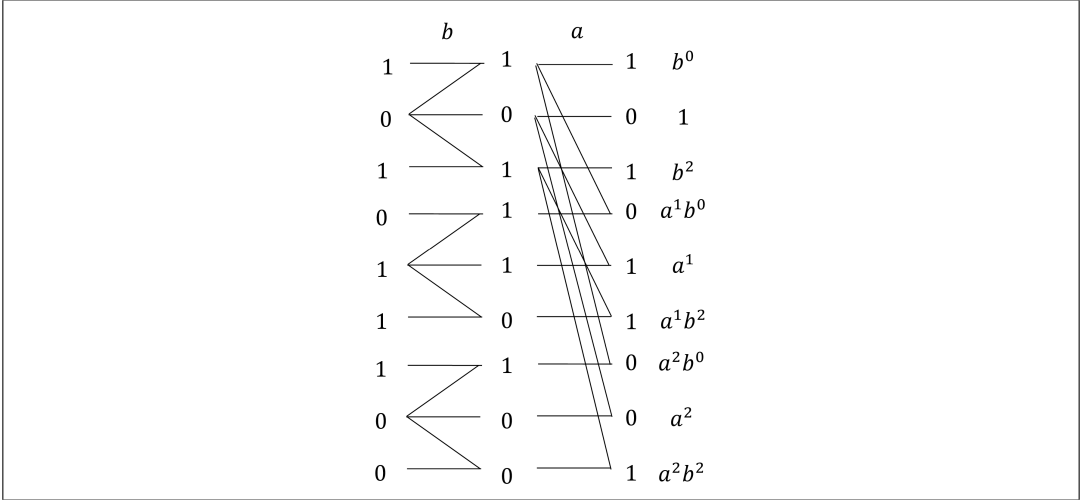


Figure 13: Butterfly for two ternary variables, with expansions with coefficient vectors $[1, A^1, A^2]$ for variable a and $[A^0, 1, A^2]$ for variable b (See Equation (20)).

Equation (20) demonstrates how to find the expression of the ternary-input form from arbitrary minterms in Figure 13:

$$F(a, b) = 1 \cdot b^0 \oplus 0 \cdot 1 \oplus 1 \cdot b^2 \oplus 0 \cdot a^1b^0 \oplus 1 \cdot a^1 \oplus 1 \cdot a^1b^2 \oplus 0 \cdot a^2b^0 \oplus 0 \cdot a^2 \oplus 1 \cdot a^2b^2$$

$$F(a, b) = b^0 \oplus b^2 \oplus a^1 \oplus a^1b^2 \oplus a^2b^2 \quad (20)$$

This expression is realized as a ternary-input binary-output reversible circuit which can also be interpreted as our learned classifier model.

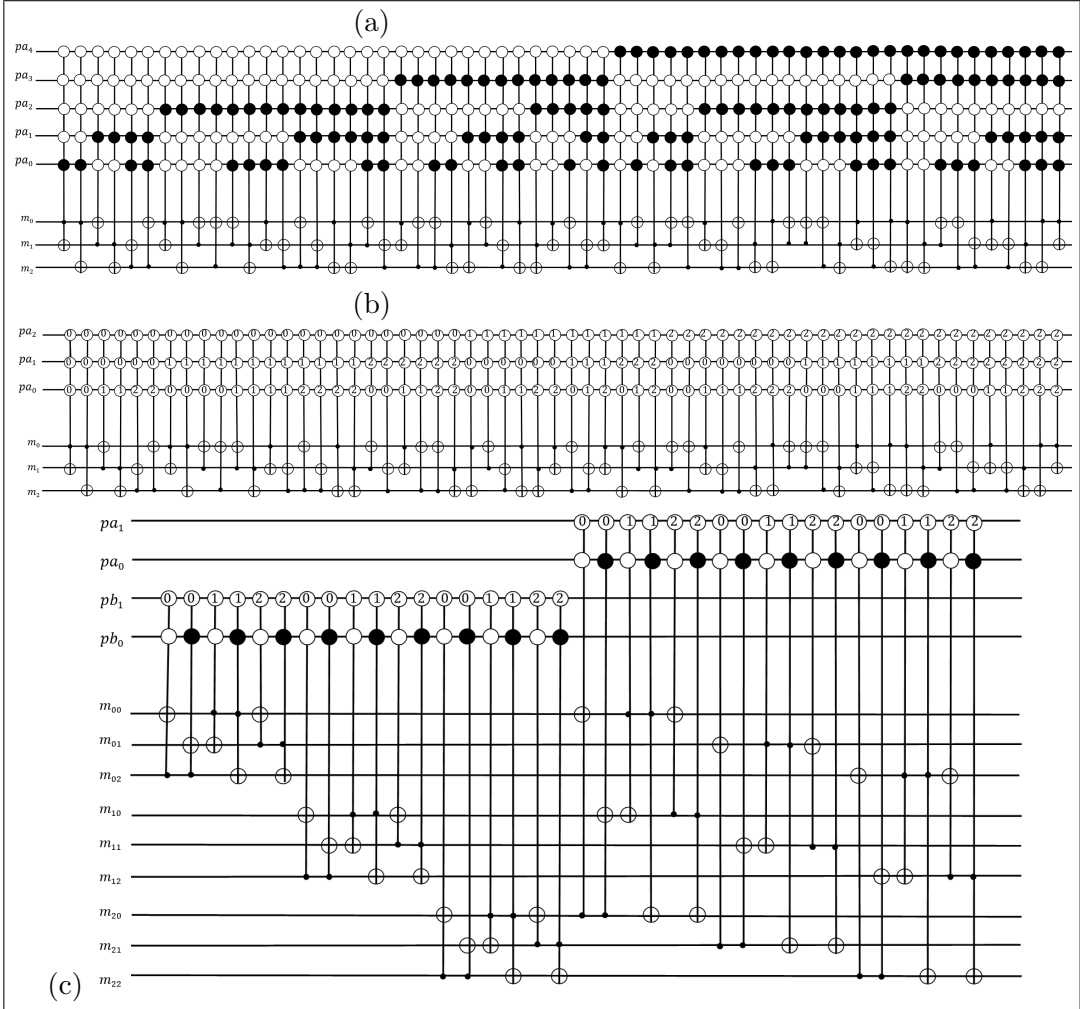


Figure 14: (a) One-variable \hat{IJKRM} multiplexer with 5 binary controls to realize all 28 ternary-input binary-output expansions, and 4 throwaway bit combinations: "11100", "11101", "11110", and "11111". By throwaway bit combination, we mean some combination of bits is not selecting any gate in our multiplexer, therefore it selects the Ternary Shannon expansion. (b) One-variable \hat{IJFPRM} multiplexer with 3 ternary controls to realize all Davio-like ternary-input, binary-output expansions. (c) Two-variable multiplexer to realize all 6 kernels from Equation (17) expansions with one ternary and one binary control for each variable.

Quantum multiplexers can be made for ternary expansions as they were made for binary in [15] and [13]. Figure 14(a) presents the kernel for a single-variable ternary

KRM and assumes Grover's Algorithm. It has 5 binary controls which create 32 combinations out of which 4 are not used to control partial kernels. These throwaway bits will be evaluated to Ternary Shannon, since its kernel indicates no gates. Figure 14(b) is for single-variable ternary FPRM and assumes ternary Grover's Algorithm. Figure 14(c) shows the butterfly for two ternary variables representing a subset of all ternary FPRM expansions— here, only 6 partial kernels are included in final one-variable kernel for simplification. Partial kernels are stacked as in Figure 6 and Figure 7, and papers [15] and [13] elaborate more on how to create butterfly circuits inside the quantum oracle.

Observe that these circuits realize reversible functions with no ancilla qudits. Therefore, to minimize such circuits, methods very similar to those from [20] can be used, however, with extending the target reversible functions from one to more qudits. Another interesting observation is this. From the perspective of circuit realization, the forms such as FPRM and KRM have no sense as ESOP is always better. They are still discussed in the literature. The same can happen in case of our hybrid forms: because of the complexity problem, maybe the simplified butterflies, as those from Figure 14(c) will still be useful, although such restricted butterflies are theoretically inferior to the full butterflies as those from Figure 14(a), Figure 14(b).

3.3 Grover's Search Algorithm and the Quantum Oracle

Grover's Algorithm is a quantum algorithm which can be used to find the input for which an unknown Boolean function outputs a 1 with $O(\sqrt{N})$ complexity given N possible elements in the solution space, whereas classical algorithms can do so with $O(N)$ complexity. A classical algorithm for this problem must check each of the N elements of the solution space (each of the 2^n minterms of the corresponding Boolean function) using a classical oracle until this oracle returns a 1. The fame of Grover's Algorithm comes from hundreds of practical and important problems that can be reduced to it— including [6, 9, 16, 17, 21]— and giving a quadratic speedup to each.

The Grover's Algorithm puts the search space into an equal superposition of all potential solutions, then increases the probability that one of the solutions (1s of the Boolean function) will be chosen. Because of space restriction we refer the reader to explanation of Grover's Algorithm and details of oracle components in [6, 9, 10, 13, 15–17, 21, 22]. The algorithm consists of three main steps. First, the solution space is initialized by creating an equal superposition of all its elements. This is done using Hadamard gates for binary and Chrestenson's gates for ternary control qudits. In general, all these MV gates can be called *Generalized Hadamards*

as their role is to create maximum equal superpositions. The qubit upon which the oracle acts is initialized to $|1\rangle$ and then it goes to a Hadamard gate, the minterms will be initialized to their corresponding values, and all other qubits initialized to $|0\rangle$. Then, the *Grover's Loop Operator*, which consists of the *Quantum Oracle* and *Diffusion Operator*, is repeated $O(\sqrt{N})$ times for problems with one solution. How we can deal with more than one solution is elaborated more in [15] and [13] and in Section 4.3. Note that the Grover's Diffusion Operator is placed only on the qubits we want to find the answer to and therefore want to measure, which are the polarity and don't know lines. Finally, the polarity control and don't know lines are measured.

The setup of the complete Grover's Algorithm for a minimum KRM of a ternary-input binary-output function with two variables is shown in Figure 15 and Figure 16.

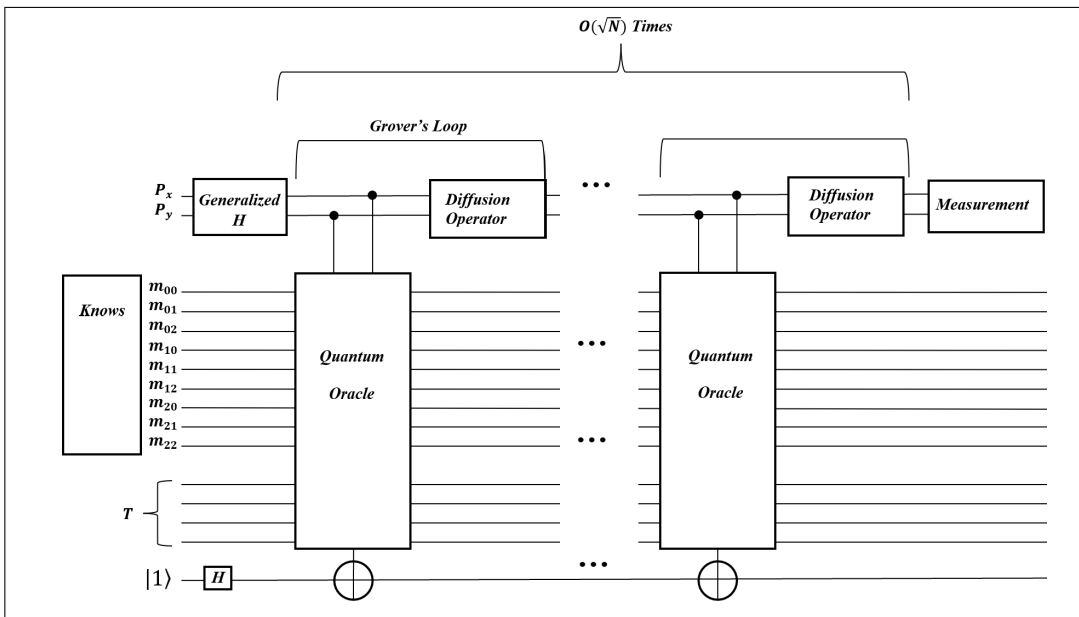


Figure 15: Block diagram of Grover's Algorithm for completely specified minterms.

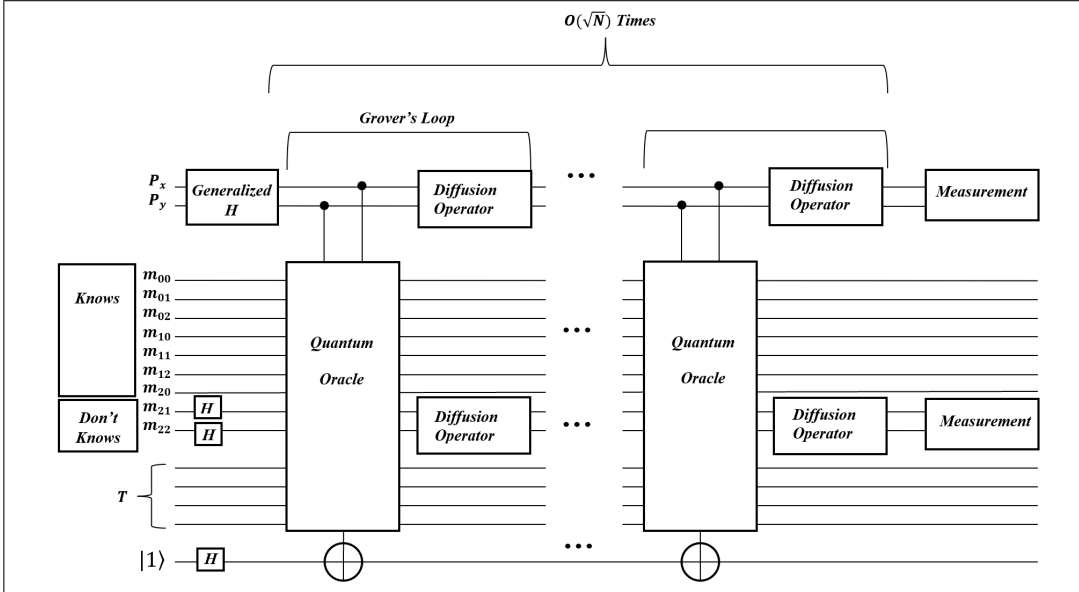


Figure 16: Block diagram of Grover's Algorithm for Machine Learning with incompletely specified minterms where m_{21} and m_{22} are don't knows. Hadamard gates are placed on m_{21} and m_{22} to indicate don't knows. Notice also the change in the diffusion operator and measurement lines.

In Figure 15 and Figure 16, P_x and P_y , which both start in $|0\rangle$, are controls for variables x and y respectively. If one wants to have exact solution for all possible expansions, these controls are 28-valued and all 28 expansion kernels are implemented in the *combined kernel* of the KRM Processor. 9 minterms are denoted as m_{00} to m_{22} at the left. These minterms are constants 0, 1, and, if we want to include Machine Learning, don't-knows. Observe that the don't-know means 0 or 1, whichever is better. A don't-know for minterm m_{jk} is naturally created for this class of algorithms as Hadamard gates in inputs m_{21} and m_{22} are placed on the left in Figure 16. *Cost Counter and Comparator* serve to remove solutions that are not equal to a selected threshold value. This way, the supervising algorithm calls Grover's Algorithm several times with modified values of the threshold, which will be described in Section 4. The Constraint Satisfaction Problems solved by Grover's Algorithm are extended to any Optimization Problems, solved by repeated use of Grover's Algorithm [9, 20].

We now present the method for the oracle of arbitrary size and type for our Machine Learning problem, which refers to the schematic in Figure 16. First, all known minterms are filled in. Don't-knows in minterms are put into equal superposition

by initializing them to $|0\rangle$, then applying Hadamard gates. Next, all polarity lines are placed into equal superposition. If these lines are multivalued, they will be put into superposition using respective Generalized Hadamard gates. The Quantum Oracle evaluates the function, reversing the phases of states that will produce function representations with spectral coefficients equal to the threshold value. The states are reversed about the mean, and the Grover's Loop Operator is repeated $O(\sqrt{N})$ times. Finally, the polarity vector and lines with don't knows are measured. In the case of binary qubits, the binary operator is used; in the case of ternary qutrits a ternary measurement operator is used [17]. Though Grover's Algorithm provides a statistically high chance of measuring a representation that is equal to the threshold value, it does not guarantee one. Because of this, the function representation found by the algorithm is checked using only the KRM Processor reversible circuit initialized to the predicted polarity and values for don't knows as found by the Grover's Algorithm. If a representation was correctly found, the classical computer controlling the quantum computer will lower the threshold value some set amount, and the Grover's Algorithm will be repeated. When no cheaper solutions are found, the last valid solution will be the exact minimal Hybrid KRM representation of the function.

Figure 17 shows the schematic for the Quantum Oracle on two ternary variables.

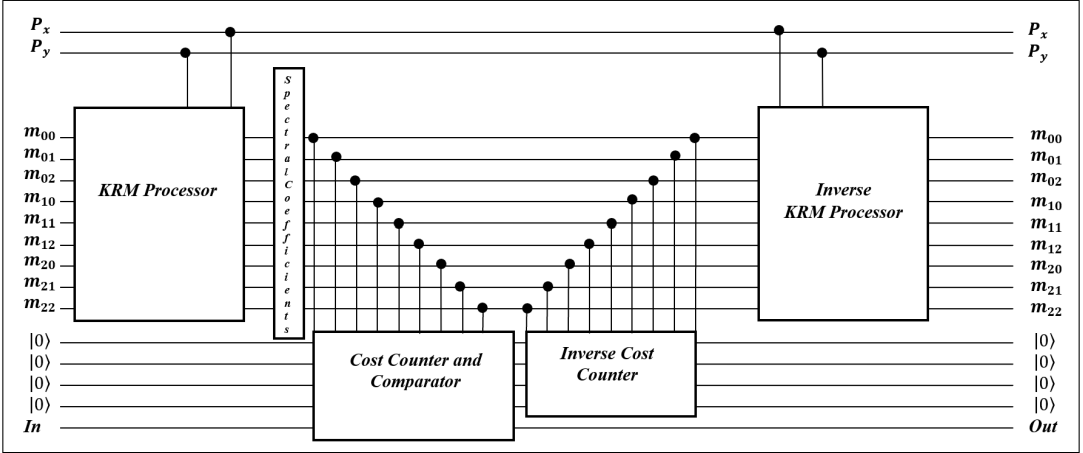


Figure 17: The schematic diagram of the Quantum Oracle for two ternary variables and 9 minterms each either initialized to 0, 1, or the superposition of 0 and 1 for a don't-know. Qudits P_x and P_y are the control lines. In includes the YES/NO function for output (Out).

Qudits P_x and P_y are hypothetically 28-valued. The Quantum Oracle checks whether a given expansion satisfies the constraint of the cost being equal to some

pre-set threshold value T . The circuit is made of the KRM Processor and its inverse, as well as the cost counter and its inverse. Each block is composed of many multi-controlled-NOT (*Generalized Toffoli*) gates. In the middle of the Quantum Oracle, between the counter and the inverse counter, there is the comparator: a Generalized Toffoli gate surrounded by inverters with its target on the *Out* qubit. Further discussion of this comparator is found in Section 4. As Grover's Loop runs the oracle many times, the inverse circuits are needed to return the input variables to their original values so that they may be used in the next iteration of the Grover's Loop. The inverse blocks, called mirrors, are created by reversing the order of the gates in the original circuits. The first block, the KRM Processor, acts on the minterms of a given function (m_{xy}), changing them to the spectral coefficients of an expansion specified by the polarity vector $[P_x, P_y]$ where the meaning of every qubit m_{00} to m_{22} is now a binary value of a corresponding spectral coefficient. The second block, the Cost Counter and Comparator, counts the number of "1" valued spectral coefficients for a given polarity and compares the number to the threshold value T . The result of this comparison is given to the output line (*Out*). If the number of 1s in the spectral coefficient values F is equal to the binary number T , this line will output a 1. Otherwise, *Out* will remain 0. For simplification, this explanation assumes the sequential creation of spectral coefficients in the oracle but of course we remember that in reality, this is done in parallel thanks to the superposition of all sets of spectral coefficients.

This optimization method is general [6, 9, 16, 17]. Grover's Algorithm as it solves decision (satisfaction) problems. Therefore, it must be called repeatedly with different threshold values to solve optimization problems. For Grover's Algorithm to solve the problem of expression minimization, it searches for an expansion equal to the threshold value, determining the cost by the number of spectral coefficient values equal to 1. If such an expansion is found, the threshold value is lowered, and the algorithm is run again. This process is repeated until no expansion is found that is cheaper than the current threshold value. The last expansion found that satisfies Grover's Algorithm is the cheapest expansion. This *quantum spectral optimization idea* can be generalized to many similar optimization problems. Unfortunately, the number of m_i qubits quickly increases with the addition of every ternary variable to the butterfly.

4 Implementation of an algorithm for the minimal ternary-input, binary-output FPRM forms and a discussion of simulated results

Here, we implement and simulate the search of the minimal ternary-input binary-output polarity on two variables. Hypothetically, the algorithm we propose works on all 28 ternary expansions. But due to the increased length, processing time, and complexity of a circuit that would encompass all ternary-input binary-output expansions, only 8 expansions were chosen to be simulated by Qiskit, a software developed by IBM specifically for quantum machines [21]. An argument can be made that, similarly as a binary FPRM does not include all the binary expansions but is still popularly used by computer scientists and engineers, selecting only some and not all of the ternary expansions in our circuit simulation, though not generating the exact minimal results, will still be sufficient for optimization. Therefore, we have programmed an algorithm that utilizes all binary controls. In addition, the method can be repeated many times with the selection of various subsets of polarity kernels in variables.

4.1 Cost counter and comparator

The purpose of the cost counter and comparator is as follows: to count the number of spectral coefficient values (counter) and compare this cost to a pre-set threshold value (comparator).

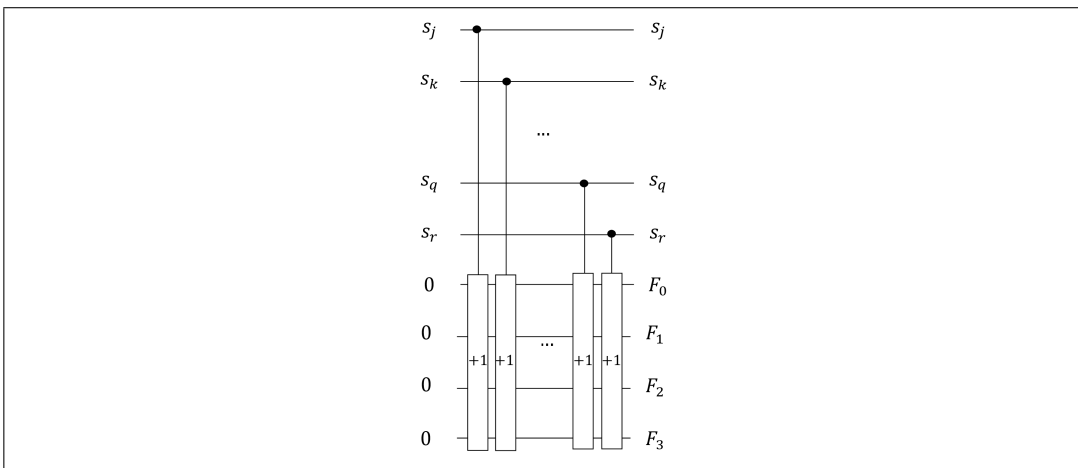


Figure 18: The schematic circuit for the counter.

4.1.1 Counter

Figure 18 reveals the schematic circuit for a 4-bit counter to be implemented after the two-variable KRM Processor. s_{j-r} are the spectral coefficient values found by the KRM Processor, $F_3F_2F_1F_0$ (F) is the (binary) number of spectral coefficient values equivalent to 1, where F_0 is the least significant bit and F_3 is the most. If the spectral coefficient value is 1, we iterate F by 1 with a (+1) gate. The (+1) gate (generalization of the Peres gate) consists of n Toffoli gates of decrementing degrees of control qubits followed by a Feynman (Controlled-NOT) gate, where n is the *size of the counter* - 1.

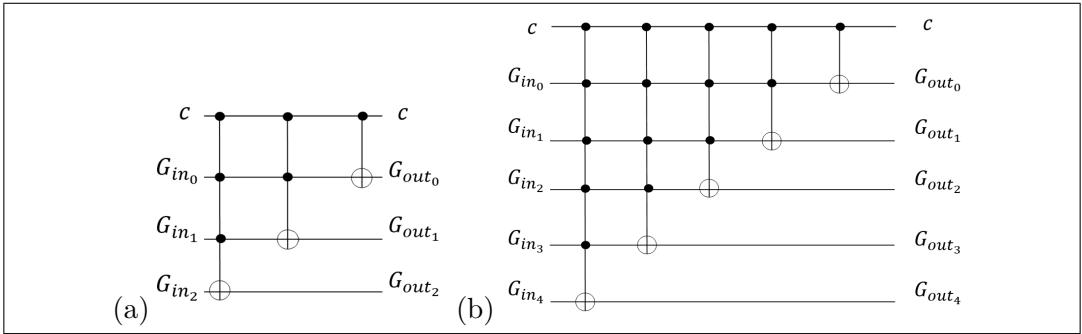


Figure 19: (a) (+1) gate for a 3-qubit counter. If the c control is 1, $G_{in_2}G_{in_1}G_{in_0}$ increments $G_{out_2}G_{out_1}G_{out_0}$ by 1. For example, if $c = 1$ and $G_{in_2}G_{in_1}G_{in_0} = 000$ then $G_{out_2}G_{out_1}G_{out_0} = 001$. (b) (+1) gate for 5-qubit counter.

As demonstrated in Figure 19, the pattern of adding Toffoli gates to increase the number of qubits can be generalized to counters of any number of qubits. The simulated oracle includes $3^2 = 9$ minterms. Therefore, we need a 4-qubit gate, but F will only reach 1001 (binary 9) if all spectral coefficient values are 1. The 4-qubit (+1) gate is controlled by iterating through the spectral coefficients as in Figure 18. Although this design uses Toffoli and Feynman gates, the optimized counter uses Peres gates built from not only CV, but also CW and other roots of NOT [2]. CV is controlled- \sqrt{NOT} .

4.1.2 Equality Comparator

The design of the comparator is as follows: to compare each qubit of F in a 4-qubit controlled Toffoli gate. NOT gates are placed where the control is 0. Figure 20 illustrates the circuit of the equality comparator of cost 1, then 2, respectively. The value we compare to F is called the threshold value.

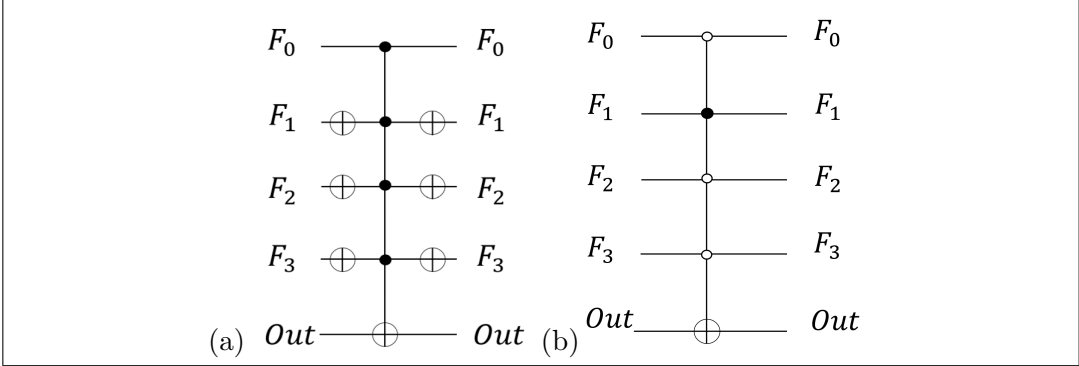


Figure 20: (a) 1-Equality Comparator (with mirrors). (b) 2-Equality Comparator (with mirrors).

Satisfaction of the comparator is evaluated on qubit Out , which in Grover's Algorithm will contribute to the negative phase. For example, if $F_3F_2F_1F_0$ is 0010, the comparator in Figure 20(a) will not be satisfied because the control qubit on F_0 of the generalized Toffoli gate will not take inputs of 0, and if we input 1 into F_1 , the NOT gate will negate 1 to 0 which also does not satisfy the generalized Toffoli. However, with the comparator in Figure 20(b), 0010 for F will satisfy the generalized Toffoli.

The threshold value is changed with each run of Grover's Algorithm. For our specific oracle of two ternary-input variables, the largest minimal cost of any possible function is 3. Therefore, we would only need to compare each set of arbitrary minterms with 1-, 2-, and 3-Equality Comparators—any more would be redundant. However, we include in Table 26 an example where this is not the case.

4.2 Qiskit KRM Processor

A fully binary oracle was simulated in language Qiskit. This oracle selects 8 Hybrid Reed-Muller expansions for every variable and has a total of 6 binary control variables for two-variable functions. The polarities are encoded into the controls of the multiplexer in correspondence to Table 21. Each variable can choose from the selection of Equation (17) polarities (6 polarities) and two Equation (14) polarities (Equation (14)a, Equation (14)b). Details of their kernels are found in Table 12.

Control Qubits (p_2, p_1, p_0)	Polarity/ Equation #
111	(17)a
110	(17)b
101	(17)c
100	(17)d
011	(17)e
010	(17)f
001	(14)a
000	(14)b

Table 21: The encoding of eight polarities of a single variable and their corresponding polarity equation numbers from Table 12.

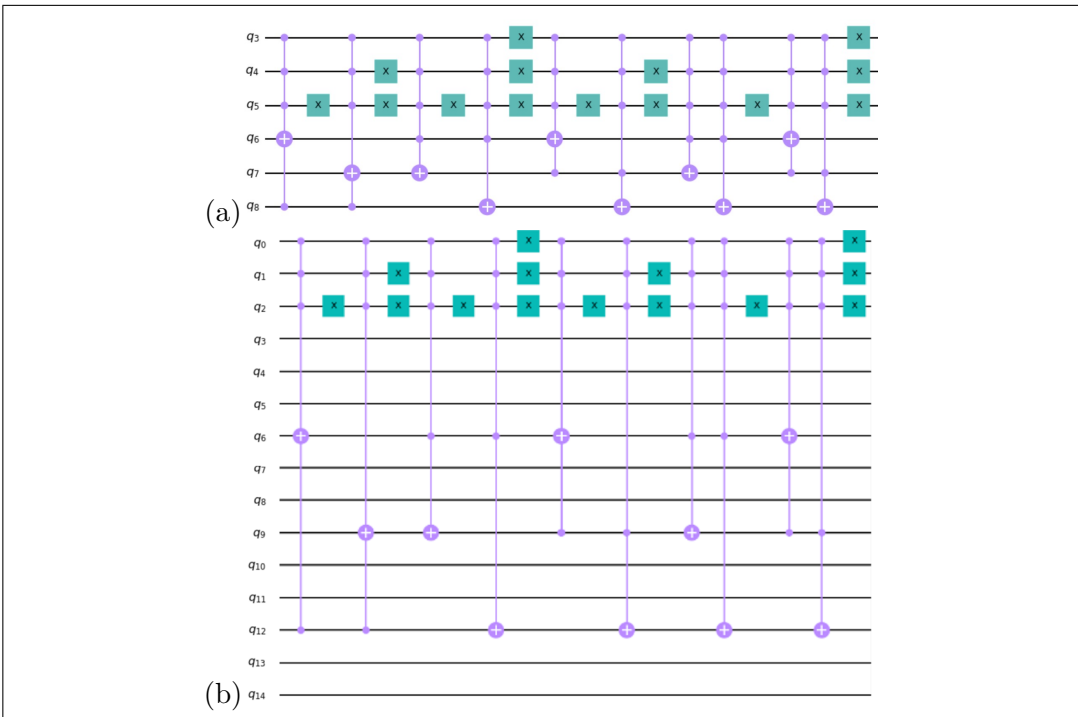


Figure 22: Drawings obtained from Qiskit software for a butterfly kernel of selected eight polarities: (a) first variable column, y ; (b) second variable column, x . These arrangements are each repeated 3 times on various qubits.

Figure 22 illustrates, in Qiskit, kernels of the multiplexer for this oracle. Figure 22(a) illustrates an unstretched kernel in the first column of the butterfly diagrams. Following the pattern of Figure 14(c), but selecting different polarities, q_3 , q_4 , and q_5 are the controls for the second variable to be expanded on, which we call y . This figure is then repeated, with q_3 , q_4 , and q_5 each three qubits larger, and repeated for a second time in the same procedure. Then, we add the second variable column as in Figure 18(b), with the same repetition as the previous kernel but this time starting on q_6 , q_9 , q_{12} . In total, the complete multiplexer/ KRM Processor uses 15 qubits, from q_0 to q_{14} .

In total our Grover's program utilizes 20 qubits: 6 for the polarities, 9 for the minterms, 4 for the counter, and 1 for the output qubit. Following the schematic design of Figure 15, Figure 16, and Figure 17, we utilize q_{15} to q_{18} (not shown in Figure 22) for the counter and comparator, and the last qubit q_{19} (not shown in Figure 22) becomes the output qubit. As it is standard in Grover's Algorithm, Hadamard gates are placed on q_0 to q_5 , and q_{19} is initialized to the $\hat{\alpha}$ state negative phase before the oracle and diffusion operators.

4.3 Qiskit Grover's Diffusion Operator and Grover's Loop

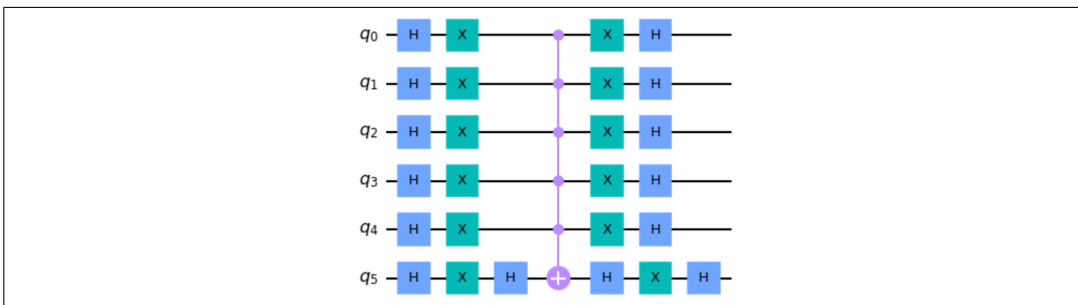


Figure 23: Grover's Diffusion Operator on six qubits.

In functions with completely specified minterms, we want to use Grover's Algorithm to find and measure polarities on q_0 to q_5 only. Therefore, standard diffusion operators will span across these qubits.

Figure 23 illustrates the diffusion oracle for this oracle. Design and discussion of how the diffusion operator works, which is not the purpose of this paper, were presented in [9, 13, 15–17, 21].

However, in this paper we introduce a new variation of the diffusion operator for Machine Learning that was not shown correctly in [13].

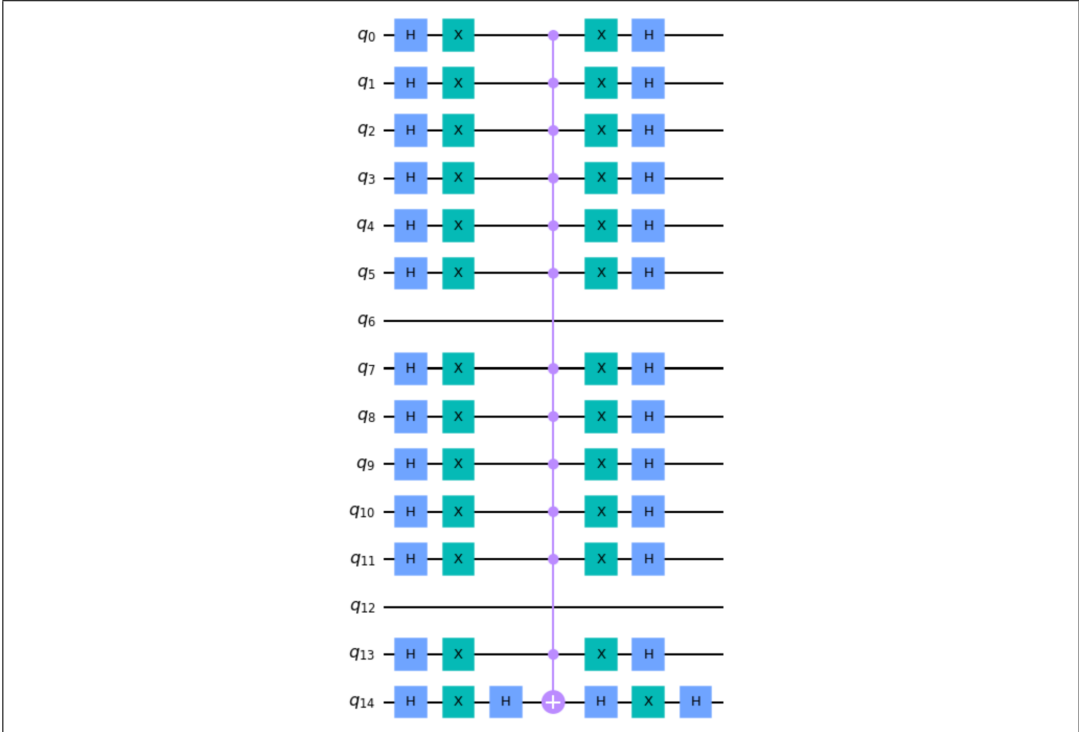


Figure 24: Grover's Diffusion Operator for minterms 1XXXXX0XX. Minterm m_{00} has a constant value 1 (qubit q_6) and m_{20} (qubit q_{12}) has a constant value 0 so the diffusion operator does not operate on the qubits q_6 and q_{12} .

In Figure 24, which corresponds to incomplete functions and thus to ML, the diffusion operator encompasses the lines with don't knows. It becomes obvious then, that the diffusion operator is unique to the minterms.

Additionally, the Grover's Loop is repeated in Grover's Algorithm as an approximation of the iteration formula $\frac{\pi}{4}\sqrt{\frac{N}{M}}$, where N is the total number of possible solutions and M is the number of correct solutions. For completely specified functions Grover's Loop number is approximated to $N = 2^6 = 64$ while assuming one solution. Therefore, for completely specified functions Grover's Loop is repeated 6–7 times. In instances of incompletely specified functions Grover's Loop is more complicated, as it is dependent on the the number of don't knows and approximate number of solutions. It is also far more difficult to evaluate the number of solutions in case of more than one solution. When determining the approximate number of repetitions of Grover's Loop for ML, we have omitted the $\frac{\pi}{4}$ from the formula as it is negligible. The method to evaluate M in this paper is good for a rather small

number of variables.

4.4 Experimental procedure

Here, we describe the procedure of collecting results. The equality comparator is pre-set first to cost 3 ($T = 3$). We input arbitrary minterms into Grover's Algorithm and set the diffusion operator in respect to the minterms. An approximate number of repetitions for Grover's Loop is calculated as well. Then, we measure the polarity and don't know lines after running the entire algorithm 1000 times. Running the algorithm many times partially solves the problem related to the unknown number of solutions. We obtain the increased probabilities of solutions. The Qiskit QASM simulator histogram displays all probabilities of each selected polarity, where it can be determined if the Grover's Algorithm is successful. If so, T is decremented, and we run the algorithm again. When we find the smallest T in which the algorithm is still successful, the polarity vector with the largest probabilistic value for 1000 repetitions of the complete Grover's Algorithm is selected. Then, the polarity and minterms (with the predicted values for the don't knows) are inputted into a separate program with just the KRM Processor, where we measure the spectral coefficient values to verify the correct polarity. Since these are small examples, we were able to verify these spectral coefficients by hand on Marquand Charts for two-variable ternary-input binary-output functions.

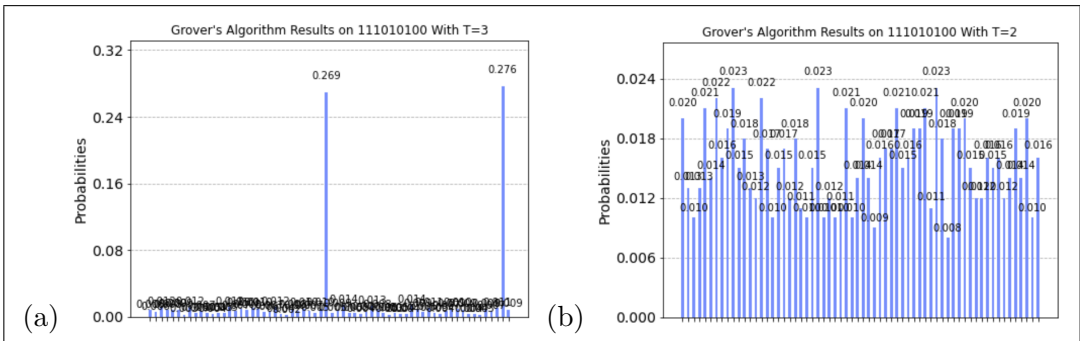


Figure 25: Qiskit histogram examples on minterms 111010100 of: (a) (Successful) Grover's Algorithm that satisfies the 3-equality comparator; (b) (Unsuccessful) Grover's Algorithm that does not satisfy the 2-equality comparator. In instances of (a) where there are multiple polarity solutions, only one is recorded.

4.5 Analysis of simulated results

Ex. #	Binary Minterms of Two Ternary Variables	Polarity Vector ($px_2 px_1 px_0$ $py_2 py_1 py_0$)	Minimum Form (spectral coefficient values vector)	Minimal # of Spectral Coefficients (from simulation)	Expression $f(x,y)$ = (spectral coefficient values · vector of base literals)
1	111111000	011000	000010000	1	x^{01}
2	011111111	000000	100010000	2	$x^0y^0 \oplus 1$
3	011011011	001010	010000000	1	y^{12}
4	010010010	001001	010000000	1	y^1
5	010101010	111111	000001010	2	$x^1y^{02} \oplus x^{02}y^1$
6	101010101	111111	000010001	2	$x^1y^1 \oplus x^{02}y^{02}$
7	111000111	111001	000000100	1	x^{02}
8	111011001	011010	100010001	3	$x^0y^0 \oplus x^{01}y^{12} \oplus x^2y^2$
9	111010100	111110	001010100	3	$x^0y^{12} \oplus x^1y^1 \oplus x^{02}y^0$
10	110111001	101101	100001001	3	$x^2y^2 \oplus x^1y^2 \oplus x^{01}y^{01}$
11	001110100	101000	001010100	3	$x^{01}y^2 \oplus x^1 \oplus x^2y^0$
12	101011011	010000	101110000	4	$x^0y^0 \oplus x^0y^2 \oplus x^{12}y^0 \oplus x^{12}$
13	110010110	100101	100010000	2	$x^{02}y^{01} \oplus x^1y^1$
14	111011111	001000	010100000	2	$1 \oplus x^1y^{01}$
15	111110011	001000	010001100	3	$1 \oplus x^2y^0 \oplus x^1y^2$
16	000111111	110000	000000010	1	x^{12}
17	000000100	000000	000000100	1	x^2y^0
18	000111000	100001	000001000	1	x^1
19	001000100	010010	001000100	2	$x^0y^2 \oplus x^2y^0$

Table 26: Minimal forms of two-variable expressions found by Qiskit for the selected 8 Hybrid Reed-Muller Forms.

Ex. #	Binary Minterms of Two Ternary Variables	Polarity ($px_2 px_1 px_0$ $py_2 py_1 py_0$)	Predicted Don't-know Value(s) X = (in respective order)	Completed Minterms as Predicted by Grover's Algorithm	Minimal # of Spectral Coefficients	Expression of $f(x,y)$ =
1	11X111001	011000	1	111111001	2	$x^{01} \oplus x^2y^2$
2	11XX11111	000001	1,1	111111111	1	1
3	111111X11	000001	1	111111111	1	1
4	00X110111	010101	0	000110111	2	$x^{12}y^{01} \oplus x^2y^2$
5	01101101X	001110	1	011011011	1	y^{12}
6	110X10110	001011	1	110110110	1	y^{01}
7	01100X000	111010	0	011000000	1	x^0y^{12}
8	00X111111	010000	0	000111111	1	x^{12}
9	1XXXXXX0XX	101001	1,1,1,1,1,0,0	1111111000	1	x^{01}
10	X1X1X00XX	011101	1,0,1,0,0	110110000	1	$x^{01}y^{01}$
11	XXXXXX111	110001	0,0,0,1,1,1	000111111	1	x^{12}
12	X1XXXXX1XX	000000	1,1,1,1,1,1,1	111111111	1	1
13	0011X1XX1	100000	1,0,0	001111001	2	$x^{02}y^2 \oplus x^1$
14	111111XXX	011001	0,0,0	1111111000	1	x^{01}
15	111111XXXX	000001	1,1,1,1	111111111	1	1
16	X1X1X0000	011101	1,0,1	110110000	1	$x^{01}y^{01}$
17	XX11XX101	001100	1,0,0,1	101101101	1	y^{02}

Table 27: Minimal form of two-variable expressions found by Qiskit for the selected 8 Hybrid Reed-Muller Forms for Machine Learning (Hadamard gates for don't-knows).

Table 26 depicts polarity and spectral coefficient results for functions with completely specified minterms, and the calculated corresponding expressions. Table 27 illustrates polarity results and the completed minterms as predicted by our algorithm. These completed minterms are not to be confused with the spectral coefficients, which are not included in Table 27. However, the spectral coefficients are obtained and used to calculate the corresponding expressions which are included in the table.

The expressions were calculated by hand by applying the given polarity to each variable as encoded (see Table 21). As described in previous sections of this paper, a tensor product of the coefficient vectors produces the symbolic names of spectral coefficients vector. These coefficient (polarity) vectors can be referred to in Table 12. Referring to Ex.2 in Table 26, the polarity is described as $\hat{A}\hat{I}J000000\hat{A}\hat{I}$, which, from Table 21 translates to Polarity Equation (14)b on variable x , and Polarity Equation (14)b on variable y . This symbolic names of spectral coefficients vector can be calculated as in Equation (21),

$$[x^0, 1, x^2] \otimes [y^0, 1, y^2] = [x^0y^0, x^0, x^0y^2, y^0, 1, y^2, x^2y^0, x^2, x^2y^2] \quad (21)$$

which, when multiplied with the spectral coefficient values vector 100010000 found through the Grover's Algorithm simulation, produces the expression $x^0y^0 \oplus 1$. We can verify this example on a Marquand Chart as depicted in Figure 28(a). As shown, the grouping satisfies all minterms of this example.

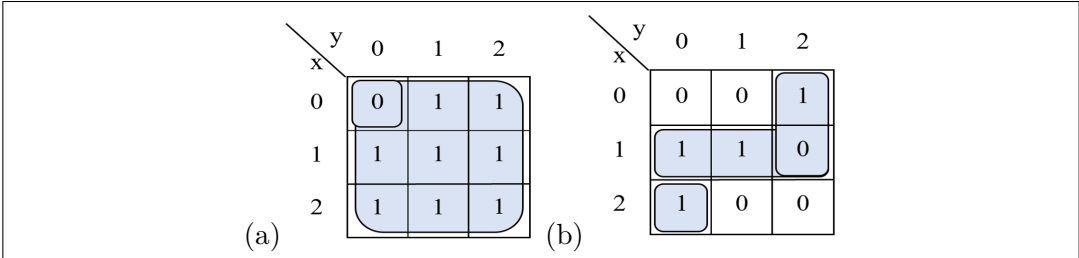


Figure 28: (a) Marquand Chart of Table 26 Ex.2 minterms with grouping $x^0y^0 \oplus 1$. (b) Marquand Chart of Table 26 Ex.11 minterms with grouping $x^0y^2 \oplus x^1 \oplus x^2y^0$.

For another example, Equation (22) calculates the vector of corresponding names of spectral coefficients from Table 26 Ex.11,

$$[x^{01}, x^1, x^2] \otimes [y^0, 1, y^2] = [x^{01}y^0, x^{01}, x^{01}y^2, x^1y^0, x^1, x^1y^2, x^2y^0, x^2, x^2y^2] \quad (22)$$

so the resulting expression becomes $x^{01}y^2 \oplus x^1 \oplus x^2y^0$. Figure 29(b) verifies this expression.

Most example solutions selected by the simulated algorithm produce the minimal number of groupings (spectral coefficients) of their minterms. However, as only 8 of the 28 possible hybrid forms were simulated in this variation of the oracle, all forms with two out of the A^{01} , A^{12} , A^{02} variable literals (present in Equation (15) and Equation (18) polarities) were excluded. This exclusivity becomes evident in Table 26 Ex.10 and 12, both of whom could have the most minimal grouping cost of 2, but instead selected 3 and 4 spectral coefficients equivalent to a value of 1, respectively. Figure 29(a) verifies the groups on Table 26 Ex.10's Marquand Chart with four spectral coefficients, while Figure 29(b) realizes these minterms with only two groups.

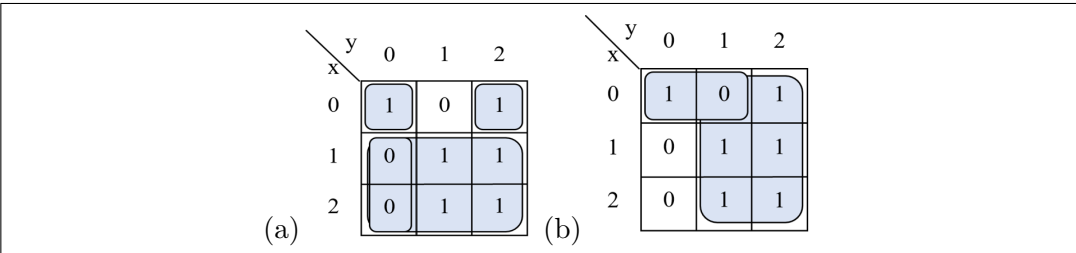


Figure 29: Marquand Chart of Table 26 Ex.10: (a) grouping $x^0y^0 \oplus x^0y^2 \oplus x^{12}y^0 \oplus x^{12}y^2$ selected by the simulated algorithm; (b) most minimal grouping with $x^0y^{01} \oplus y^{12}$ (which is obviously better than the grouping in (a)).

In ML instances, these groups are used to make predictions for don't-knows. Figure 30 illustrates Table 27 Ex.7, a mostly completely specified set of minterms. Our Grover's Algorithm can produce distinguishable polarity results that, when verified by the KRM Processor, yields at least one solution of spectral coefficients that corresponds to T . A solution given by the simulation is grouped on Marquand Charts in Figure 30.

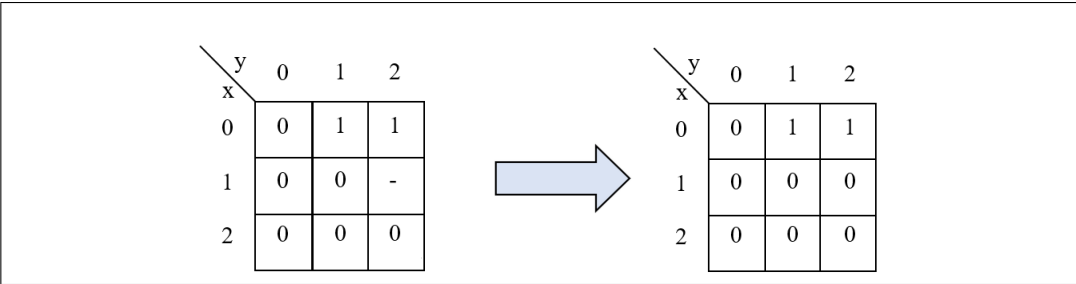


Figure 30: Marquand Charts of the Table 27 Ex.7 quantum ML prediction given by Grover's Algorithm.

Through the ML process, our incomplete minterms become completely specified. To calculate the expression and find the groupings, we place the completely specified minterms the KRM Processor once more to obtain the spectral coefficients values. Figure 31 illustrates the grouping to Table 27 Ex.7 as x^0y^{12} .

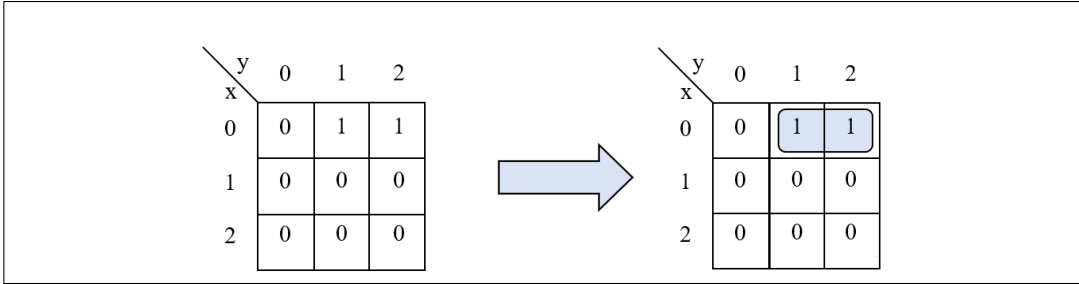


Figure 31: Marquand Charts with grouping x^0y^{12} of the quantum ML predictions for Table 27 Ex.7.

It is determined that the quantum ML method on Grover's Algorithm proposed by [13] creates accurate cost restriction results only for nearly completely specified discrete functions. Thus our algorithm is an improvement from [13] because it creates accurate cost-restricted predictions for any number of don't knows in our minterms. The difference is that our diffusion operator encompasses the don't knows, and we measure the don't know lines along with the polarity to obtain the predicted values. The simple rule for this new type of Grover's Algorithm is the following: for any unknown piece of data there must be a qubit for a don't care which means a Hadamard gate initialized to $|0\rangle$, a diffusion qubit in a diffusion circuit (as in Figure 24), and a respective measurement gate.

As an example of a highly incomplete set of minterms, refer to Table 27 Ex.9 with minterms 1XXXXX0XX.

The minterms are inputs to Grover's Algorithm with Hadamard gates placed to indicate don't knows, the threshold value T is set to 1, the diffusion operator is drawn as in Figure 23, and Grover's Loop is repeated 37 times after approximating that $N = 2^{13}$ and $M = 6$. After running the algorithm and measuring the polarity and don't know lines 1000 times, the algorithm produces the histogram shown in Figure 32.

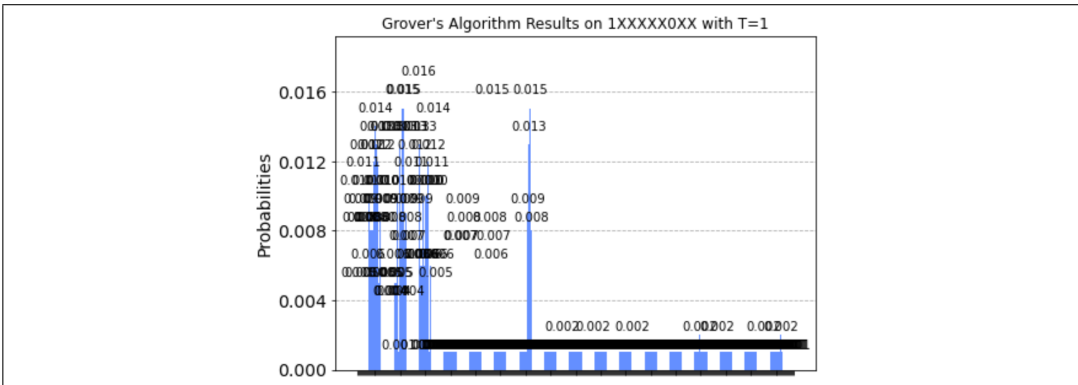


Figure 32: Probability histogram of 1000 runs of Grover's Algorithm on Table 27 Ex.9 with $T = 1$ and Grover's Loop repeated 37 times.

As expected for a large number of don't knows, there are many possible solutions even with the limited 8 expansions we programmed into our oracle. As is done with completely specified minterms, the set of polarity and don't know values with the largest probability as shown in the histogram is selected, and verified by Marquand Charts. However, for this example we have verified the many other solutions as good predictions to the don't knows as well. Since Figure 32 is extremely condensed and difficult to read, we have taken all the solutions that out of the 100 runs of Grover's Algorithm were selected more than 10 times and placed them in a separate probability histogram in Figure 33.

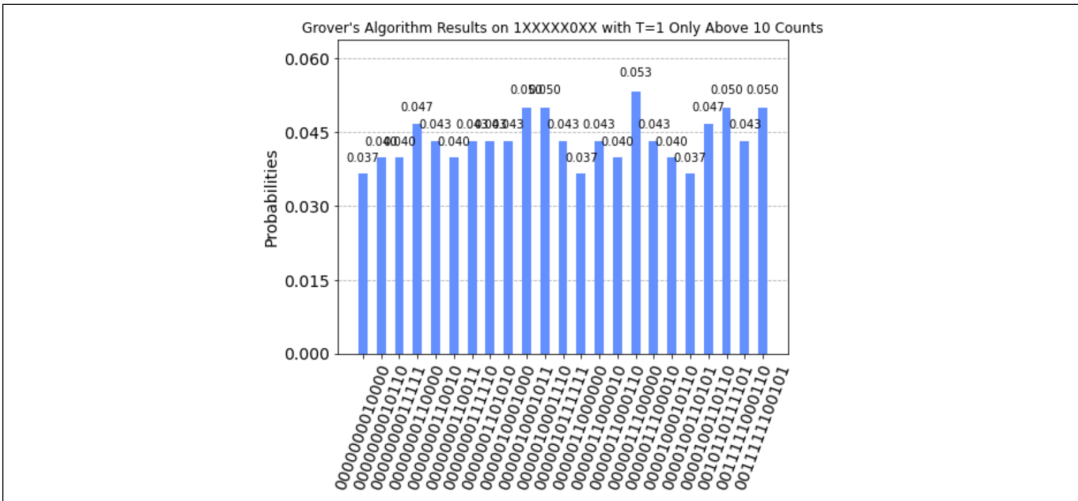


Figure 33: Probability histogram on Table 27 Ex.9 with correct solutions only.

Figure 33 shows the results of this example with solutions that were selected greater than 10 times out of 1000 total runs. Qubits are measured in reverse order, so for the rightmost probability bar, 101001 will be the polarity vector and 1111100 will be the predicted values of the don't knows in respective order to the index of the minterms. Thus the Marquand Chart is filled with the predicted values as in Figure 34.

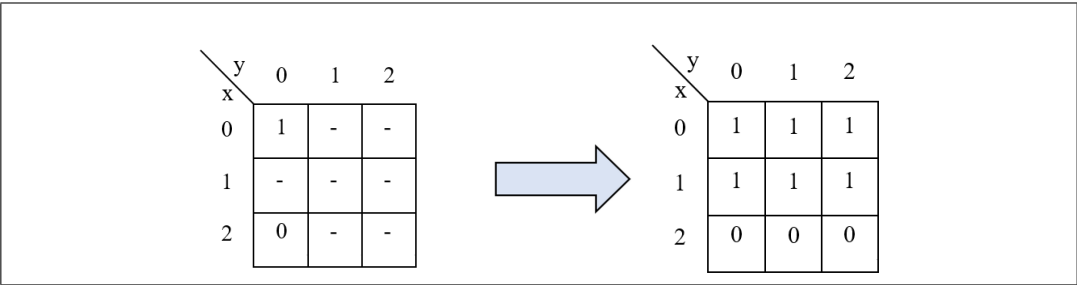


Figure 34: Marquand Chart of Table 27 Ex.9 with quantum ML 1111100 predictions from Grover’s Algorithm.

Placing these minterms into the KRM Processor, we obtain the vector of spectral coefficient values 100000000, so therefore the grouping is x^{01} . Figure 35 shows the grouping to the minterms with the predicted values.

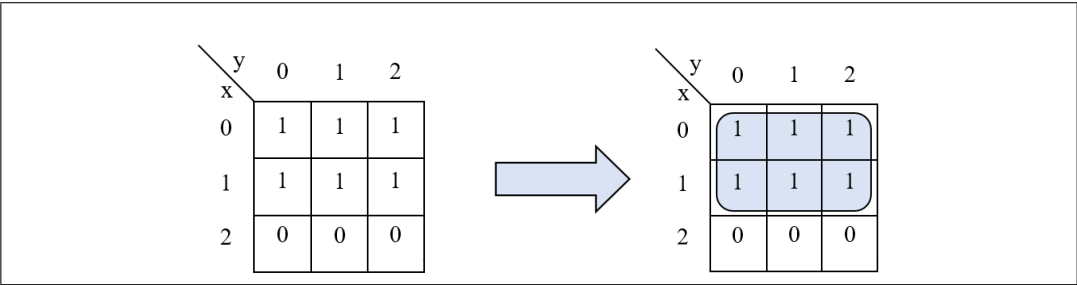


Figure 35: Marquand Chart with grouping x^{01} of quantum ML predictions for Table 27 Ex.9.

Finally, some results have the same \hat{cost} as we have determined in this paper to be the number of spectral coefficients with value 1, but the expression has a different number of literals and therefore also complicates cost calculation. Future optimization research includes designing a counter and comparator that determines the cost in respect to the number of literals.

Other variants will use a comparator of order (such as less than) rather than a

comparator of equality which will allow us to reduce the number of repetitions of Grover's Algorithm in an optimization loop.

5 Conclusion

Papers [15] and [13] presented methods to realize quantum algorithms to find exact solutions to binary FPRM and KRM minimization problems. In this paper, we formulated equivalent problems with ternary inputs and presented methods to build Grover's oracles for them. Grover's Algorithm gives quadratic speedup. When there are many solutions, the efficiency of Grover's Algorithm improves. With the arrival of quantum computers with more qudits, the results of this paper will become more practical. Because our future circuits will also include ternary qutrits, we will be not able to simulate them in Qiskit. Therefore, we will work on a GNU Octave (MATLAB) simulator for such circuits.

Even some of our oracles discussed in this paper assume ternary controls [17]. Ternary circuits are already possible with existing quantum realization technologies. Although our method can currently be applied only to functions with 2 variables in language Qiskit [21], with the development of quantum computers of more qubits, as well as with qudits that will have more values, the multi-valued variants of the presented method will become more practical. It is worthy to add that building ternary quantum circuits based on qutrits is already possible and thus ternary quantum computers will perhaps appear some day. In the past, the only practically realized multi-valued computers were the two ternary classical computers built in Russia [3]. Synthesizing ternary quantum circuits using quantum computers, as proposed in a very preliminary way in this paper, will hopefully be used in the future to design ternary quantum computers. Another interesting aspect of our work is a natural way of dealing with don't-knows: using our trick with superpositions for don't cares and don't knows is possible in quantum algorithms and rather difficult to program in classical algorithms.

In the future, we will work on other minimization methods that will make use of Grover's oracles based on butterflies for other spectral transforms, such as Pseudo-Kronecker Reed-Muller [4, 12]. The open problem is how to optimally encode hypothetical 28-valued variables to sets of small arity qudits. We were not able to find papers about quantum circuit encoding, although there are plenty of papers about various encoding problems for binary classical circuits. The only available paper is [6], but this paper only compares various encodings for Grover's Algorithm and does not create a good encoding method.

References

- [1] C. Barbieri and C. Moraga. Cycles-based and Transformation-based Synthesis of Ternary Reversible Circuits. Aspects of Complexity. *Journal of Applied Logics*, 8:1295–1309, 2021.
- [2] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter. Elementary Gates for Quantum Computation. *Phys. Rev. A*, 52:3457–3467, Nov. 1995.
- [3] N. P. Brousentsov, S. P. Maslov, A. J. Ramil, and E. A. Zhogolev. Development of Ternary Computers at Moscow State University, 1997.
- [4] M. Chrzanowska-Jeske, A. Mishchenko, and M. Perkowski. Generalized Inclusive Form-sâŤNew Canonical Reed-Muller Forms Including Minimum ESOPs. *VLSI Design*, 14:13–21, Feb. 2002.
- [5] P. Clark and T Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3:261–283, 1989.
- [6] S. Dhawan and M. Perkowski. Comparison of Influence of Two Data-Encoding Methods for Grover Algorithm on Quantum Costs. In *2011 41st International Symposium on Multiple-Valued Logic*, pages 176–181, 2011.
- [7] J.W. Eaton. GNU Octave, 2022.
- [8] P. Gao, Y. Li, M. Perkowski, and X. Song. Novel Quantum Algorithms to Minimize Switching Functions Based on Graph Partitions. *CMC-Computers, Americals and Continua*, accepted.
- [9] P. Gao, Y. Li, M. Perkowski, and X. Song. Realization of Quantum Oracles using Symmetries of Boolean Functions. *Quantum Inf. Comput.*, 20:418–448, 2020.
- [10] L. K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC 96*, pages 212–219, 1996.
- [11] The MathWorks Inc. MATLAB, 2022.
- [12] K. Jin, T. Saffat, J. Morgan, and M. Perkowski. A Polarity-based Approach for Optimization of Multivalued Quantum Multiplexers with Arbitrary Single-qubit Target Gates. *FLAP*, 7:5–28, 2020.
- [13] B. Lee and M. Perkowski. Quantum Machine Learning Based on Minimizing Kronecker-Reed-Muller Forms and Grover Search Algorithm with Hybrid Oracles. In *2016 Euromicro Conference on Digital System Design (DSD)*, pages 413–422, 2016.
- [14] T. Lewis, M. Perkowski, and L. Jozwiak. Learning in Hardware: Architecture and Implementation of an FPGA-Based Rough Set Machine. In *Proceedings of the Euro-Micro’99 Conference*, pages 1326–1334, Milano, Italy, Sep. 1999.
- [15] L. Li, M. Thornton, and M. Perkowski. A Quantum CAD Accelerator Based on Grover’s Algorithm for Finding the Minimum Fixed Polarity Reed-Muller Form. In *36th International Symposium on Multiple-Valued Logic (ISMVL’06)*, pages 33–38, May 2006.
- [16] Y. Li, E. Tsai, M. Perkowski, and X. Song. Grover-based Ashenhurst-Curtis Decomposi-

- tion Using Quantum Language Quipper. *Quantum Info. Comput.*, 19(1&2):35&66, 2019.
- [17] S. B. Mandal, A. Chakrabarti, and S. Sur-Kolay. Synthesis of Ternary Grover’s Algorithm. In *2014 IEEE 44th International Symposium on Multiple-Valued Logic*, pages 184–189, 2014.
 - [18] R.S. Michalski. On the Quasi-Optimal Solution of the General Covering Problem. In *Proceedings of the V International Symposium on Information Processing (FCIP 69)*, volume A3, pages 125–128, Bled, Yugoslavia, 1969.
 - [19] A. Mishchenko, C. Files, M. Perkowski, B. Steinbach, and C. Dorotska. Implicit Algorithms for Multi-Valued Input Support Manipulation. In *the 4th International Workshop on Boolean Problems*, Sep. 2000.
 - [20] M. Perkowski. Inverse Problems, Constraint Satisfaction, Reversible Logic, Invertible Logic and Grover Quantum Oracles for Practical Problems. In *LNCS*, volume 12227, pages 3–32, July 2020.
 - [21] IBM Research. Qiskit, 2017.
 - [22] T. Sasao and J.T. Butler. A Design Method for Look-up Table Type FPGA by Pseudo-Kronecker Expansion. In *Proceedings of 24th International Symposium on Multiple-Valued Logic (ISMVL’94)*, pages 97–106, 1994.
 - [23] N. Song and M. Perkowski. Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input, Incompletely Specified Functions. *IEEE Transactions on Computer Aided Design*, 15:385–395, Apr. 1996.

DNA TECHNOLOGY FOR MULTI-VALUED DATA STORAGE USING READ ONLY MEMORY

HAFIZ MD. HASAN BABU^{1*}, KHANDAKER MOHAMMAD MOHI UDDIN² ,
TAMANNA TABASSUM³ , MOHAMMED NASIR UDDIN⁴

¹ *Department of Computer Science and Engineering, University of Dhaka,
Dhaka-1000, Bangladesh*

² *Department of Computer Science and Engineering, Dhaka International
University, Dhaka-1205, Bangladesh*

^{3,4} *Department of Computer Science and Engineering, Jagannath University,
Dhaka-1100, Bangladesh*

`hafizbabu@du.ac.bd`

Abstract

DNA (Deoxyribose Nucleic Acid) computing has the features of parallel processing and large storage capability that make it special from other conventional computing systems. It is a type of biomolecular programming where different types of reactions are used to perform basic operations and the processing information is stored in nucleic acids and proteins. The traditional ROM (Read Only Memory) is a slower memory. Thus, the multi-valued DNA computing enables the creation of new types of computers which is capable of operating multiple sequences as input states by increasing storage capacity. In this paper, a multi-valued DNA-based ROM architecture is designed using proposed algorithms of multi-valued DNA-based operations.

keywords : DNA computing, parallel processing, Biomolecular programming, ROM architecture, Multi-valued.

1 Introduction

Modern computers work on the basis of binary logic where everything is represented using 0 and 1. Boolean gates take input as 0 or 1 and generate output in a binary form. signal transduction uses the principle of binary logic which is controlled by molecular switches [1]. Though binary logic uses just two logics true or false,

*Corresponding author. E-mail address: hafizbabu@du.ac.bd

sometimes uncertainty and imprecision are occurred by the logic gates. To get rid of these problems, multi-valued logic is introduced where switching can be possible in more than two states. Besides, higher information densities are allowed by the multi-valued logic using the increasing number of distinguishable states [2].

Different types of molecular computing systems are introduced using DNA computational systems which may work with or without enzymes [3]. The genetic information of biological organisms is encoded by DNA (deoxyribonucleic acid) [4]. It is made up of polymer chains, often known as DNA strands. Each strand is made up of nucleotides, or bases, that are connected to a sugarphosphate “backbone”. Adenine, guanine, cytosine, and thymine are the four DNA nucleotides, abbreviated as A, G, C, and T, respectively. The 5' end of one sequence pairs with the 3' end of the corresponding sequence in an antiparallel fashion in DNA. The reverse complement is seen right to left when complementary sequences are written in the 5' -> 3' manner. The capacity to generate short DNA sequences artificially allows these sequences to be used as inputs for algorithms. DNA has properties that allow it to be used to simulate classical logic processes. DNA prefers to be in double stranded form, while single stranded DNA naturally migrate towards complementary sequences to form double stranded complexes.

In 1994, Leonard Adleman developed the first DNA computer prototype. This developed prototype can solve the Hamiltonian path problem using DNA polymerase and ligase [5].

After that restriction toe-hold exchange [6], endonuclease [7], [8], deoxyribose [9] have been used to develop DNA-based computing machines. The toe-hold exchange plays a promising role to perform DNA computing which is based on a seesaw gate motif [10]. In neural network computations, DNA seesaw circuits have been used [11], [12]. DNA computing has some good properties which make it more useful and more efficient than traditional computing systems [13].

The merits of DNA computing are given as follows:

- Vast parallelism is the key feature of DNA computing that enables it to perform millions of operations at a time [14]. If we make it operational sequentially then it will lose its edge [15].
- DNA could store data for a long time in which data storage capacity is also gigantic [16].
- DNA computing works on base-pair computing which provides unique error correction techniques like the RAID 1 array [17].
- Unless it faces harsh, DNA is a stable molecule. The mutation is quite difficult in DNA [18].

A number of works based on DNA and molecule computing have been done at present time. Gupta et al. [19] introduced a DNA-based model to perform logical and arithmetic operations. Operations execute in this model in linear or serial-parallel models. Based on deoxyribozyme-based logic gates, Stojanovic et al. [9] proposed a half adder logic circuit. On the other hand, Lederman [20] designed a full adder using DNA basic gates where three oligonucleotides are used as inputs, and the final outputs are obtained after the reactions of fluorogenic-cutting. A half subtractor has been developed by Pe'rez-Inestrosa et al. [21] utilizing the combination of INHIBIT and XOR gates.

Continuous technological advancement – driven by Moore's Law - electronic gadgets and their linked memory systems have been able toward becoming simultaneously smaller and more efficient. Current storage capacity and other cognitive mechanisms on the other hand consume more energy. Furthermore, internal thermal resistance and other restrictions may hinder further memory storage advancements even as worldwide demand for enhanced data storage and retrieval capabilities continues to expand. So, the main concern is to provide low-cost, robust, high-density, reliable, and energy-efficient memory technologies through designing multi-valued DNA-based ROM. This technology has the ability to be written on, read from, and erased rapidly, and not deteriorate with time.

2 Background Study

In this section, the basics of multi-valued computing, multi-valued DNA computing, multi-valued DNA basic gate operations with algorithms, and multi-valued DNA ROM are discussed in detail.

2.1 Multiple-Valued Basic Operations

In ternary logic [22], [23], a third value is acquainted with binary logic. In this paper, false, undefined, and true are defined as 0, 1, and 2, respectively [24].

The basic operations of ternary logic can be defined as follows [25], [26]:

$$Y_{OR} = \max(x, y) \tag{1}$$

$$Y_{NOR} = \overline{\max(x, y)} \tag{2}$$

$$Y_{AND} = \min(x, y) \tag{3}$$

$$Y_{NAND} = \overline{\min(x, y)} \tag{4}$$

$$Y_{XOR} = diff(x, y) \tag{5}$$

$$Y_{XNOR} = \overline{diff(x,y)} \quad (6)$$

Where, $x,y=0,1,2$

The truth table of Ternary AND, NAND, OR, NOR, XOR, and XNOR logic gates for ternary logic operations presents in Table 1. For AND logic gate its output value depends on the minimum value of its inputs. Similarly, in the case of the OR logic gate, its output value depends on the maximum value of its inputs. For the XOR gate, its output value is the difference of the value of its inputs. Finally, the outputs of NAND, NOR, and XNOR logic gates become the inverted of AND, OR and XNOR logic gates.

Input1	Input2	AND	NAND	OR	NOR	XOR	XNOR
0	0	0	2	0	2	0	2
0	1	0	2	1	1	1	1
0	2	0	2	2	0	2	0
1	0	0	2	1	1	1	1
1	1	1	1	1	1	0	2
1	2	1	1	2	0	1	1
2	0	0	2	2	0	2	0
2	1	1	1	2	0	1	1
2	2	2	0	2	0	0	2

Table 1: Truth Table for Ternary AND, NAND, OR, NOR, XOR, and XNOR

2.2 Multiple-Valued DNA Computing

A ternary or three-valued logic function is one that has two inputs that can assume three states (say 0, 1 and 2) and generates one output signal that can have one of these three states. Two DNA sequences are utilized as inputs and one DNA sequence is used as output in ternary DNA computing. The sequence ACCTAG is considered as “0,” the sequence CAAGCT strands as “1,” and TGGATC as “2” in this proposed ternary DNA computing.

In this Ternary DNA computing, the fluorescent level is used to detect the DNA sequence. Fluorescence is the temporary absorption of electromagnetic wavelengths from the visible light spectrum by fluorescent molecules, and the subsequent emission of light at a lower energy level. When it occurs in a living organism, it is sometimes called biofluorescence. This causes the light that is emitted to be a different color than the light that is absorbed. Stimulating light excites an electron, raising energy to an unstable level.

Fluorescent dyes can attach to nucleotide sequences via the sugar ring, the phosphate backbone, or the nucleotide itself. To identify the sequence composition, the sequence must be run through a laser that can differentiate each of the fluorescently labeled nucleotide bases in a chromatogram. A chromatogram is a graph of each component's intensity as a function of time. As a result, one fluorescent color will be high intensity at each place in the sequence, while the other three fluorescent colors will be low intensity. The high intensity colors in a chromatogram, for example, are red, black, red, red, green, red, blue, blue, black, blue, which correlates to the nucleotide sequence TGGTATCCGC[27].

2.3 Multiple-Valued DNA Basic Operations

In this section, multi-valued DNA basic gates are explained with the algorithm.

2.3.1 DNA Ternary Inverter

In a Ternary Inverter or NOT gate, it obtains an inverted output of its input value. The design procedure of DNA Ternary NOT Gate is quite easy. DNA Ternary NOT Gate or inverter is illustrated in Figure 1. During DNA Ternary computing, sequence ACCTAG is considered as “0”, sequence CAAGCT strands as “1” and TGGATC stands as “2”. So, for doing NOT operation in DNA Ternary Inverter, one of the sequences is inserted into the tube. The fluorescent level is used here to detect the sequences and logical value of the sequences. Finally, this proposed DNA Ternary Gate gives the inverted output of the input sequence.

Based on the operating principle, a General Ternary Inverter (GTI) can be of three types: Negative, Positive, and Standard. A GTI is represented by Equation (1), Equation (2) and Equation (3), where x is the input and y_0 , y_1 and y_2 are the outputs that represent a Negative Ternary Inverter (NTI), a Positive Ternary Inverter (PTI) and a Standard Ternary Inverter (STI), respectively [28]. The truth table that represents the functions y_0 , y_1 , and y_2 which is shown in Table 2. Our work is based on Standard Ternary Inverter (STI). Table 3 shows the truth table

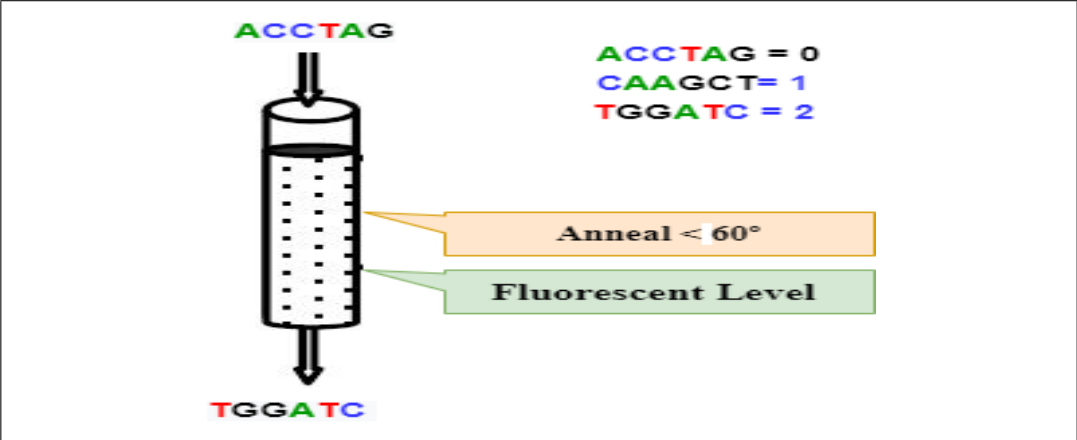


Figure 1: Illustration of a DNA Ternary NOT Operation.

of DNA Ternary NOT Gate (STI). The fluorescent level is used here to detect the sequences and logical value of the sequences.

$$y0 = \int_{0,x \neq 0}^{2,x=0} \tag{7}$$

$$y1 = \int_{0,x=2}^{2,x \neq 2} \tag{8}$$

$$y2 = x' = 2 - x \tag{9}$$

Input 1	NTI(y0)	PTI(y1)	STI(y2)
0	2	2	2
1	0	2	1
2	0	0	0

Table 2: Truth Table representing NTI, PTI and STI

In Table 3, when input sequence is ACCTAG, the gate generates the output TGGATC. If the input sequence is CAAGCT, then the gate generates the output CAAGCT. When the input sequence is TGGATC, the gate generates the output ACCTAG. Table 3: Truth Table of DNA Ternary NOT Gate (STI)

Input 1	Output
ACCTAG	TGGATC
CAAGCT	CAAGCT
TGGATC	ACCTAG

Table 3: Truth Table of DNA Ternary NOT Gate (STI)

2.3.2 DNA Ternary AND Operation

DNA Ternary AND gate consists of two ternary inputs and one ternary output. A DNA Ternary AND gate is obtained using two inputs and one output. The lower logical sequence value from the input sequences is considered as the output of DNA Ternary AND Gate and when the input sequences have the same logical value, one of them is considered as the output. The fluorescent level is used here to detect the sequences and logical value of the sequences. Figure 2 shows the DNA Ternary AND gate operation.

Ternary AND function is also similar to the Binary “MINIMUM OF” function. The Algorithm 1 shows the working procedure of DNA multi-valued AND operation. The following equation helps to get the DNA AND gate output from the ternary inputs:

$$Y_{AND} = \min(x, y) \quad (10)$$

2.4 Read Only Memory

Read-only memory (ROM) is a type of non-volatile memory used in computers and other electronic devices. Data stored in ROM cannot be electronically modified after the manufacture of the memory device. It's used to store the start-up instructions for a computer, also known as the firmware. Most modern computers use flash-based ROM. It is part of the BIOS chip, which is located on the motherboard. Obtain the function output F1 and F2 in sum of minterms form, F1 and F2 = $\Sigma (0, 1, 2, 3, 4, 5, 6, 7, 8)$ [29]. An architecture of a ROM is shown in Figure 3 and Figure 4 shows the circuit diagram of a ROM.

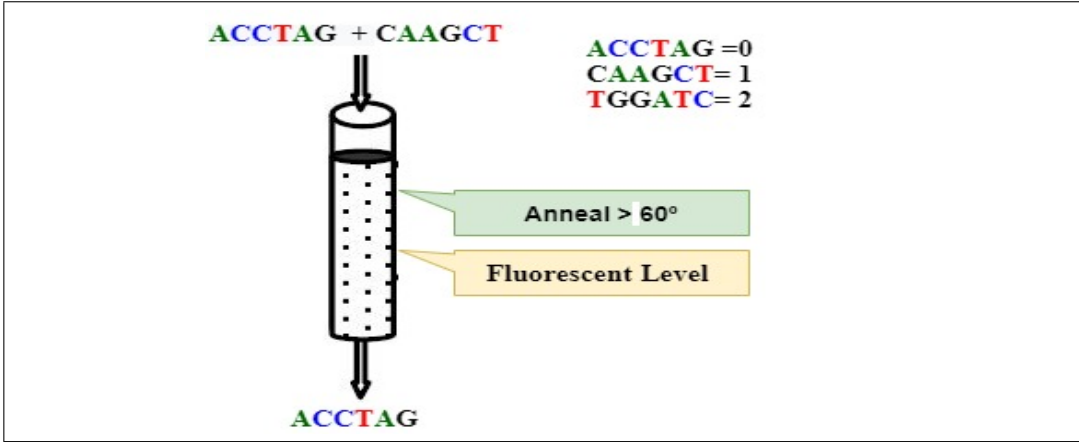


Figure 2: Illustration of a DNA Ternary AND Operation.

Algorithm 1 Multiple-Valued DNA-based AND Gate

Input: X, Y
Output: ACCTAG, CAAGCT, TGGATC
Begin
procedure *DO_MV_DNA_AND*(X, Y)
 if *X equals Y* **then**
 Do_Result(X);
 else if *X < Y* **then**
 Do_Result(X);
 else if *X > Y* **then**
 Do_Result(Y);
 end if
end procedure
procedure *Do_Result*(Q)
 $P \leftarrow \text{fluorescent_get_value}(Q)$;
 if *P equals 0* **then**
 return ACCTAG
 else if *P equals 1* **then**
 return CAAGCT
 else if *P equals 2* **then**
 return TGGATC
 end if
end procedure
End

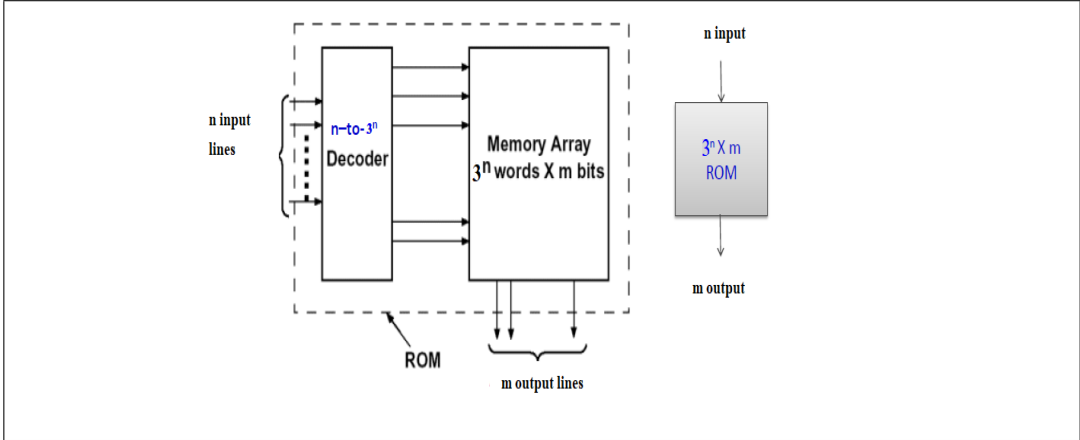


Figure 3: ROM Architecture.

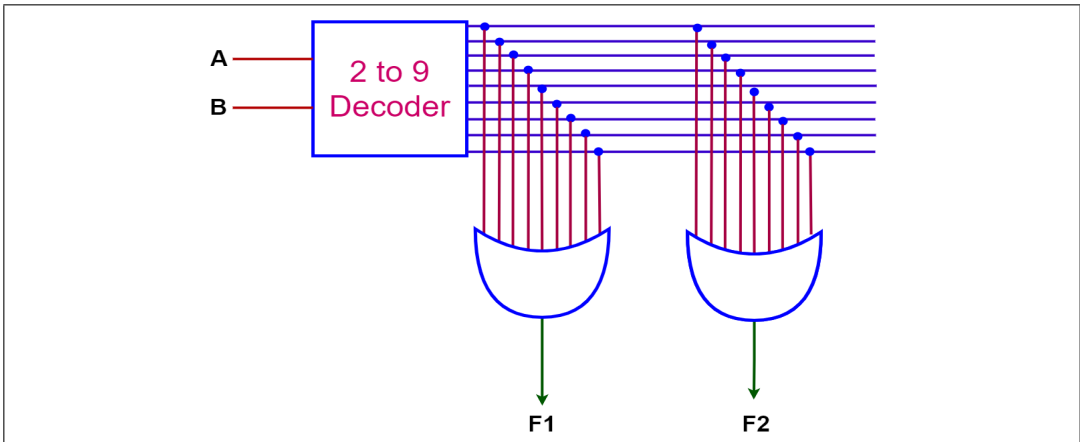


Figure 4: Circuit Diagram of ROM Architecture.

3 Multi-Valued ROM Design Using DNA Computing

In this 9 to 2 multivalued DNA based ROM has two inputs DNA sequences A and B with three states A (ACCTAG = 0, CAAGCT = 1, TGGATC = 2) and B (ACCTAG = 0, CAAGCT = 1, TGGATC = 2) and two output functions F1 and F2. Though 2 inputs can take 9 values and produce 2 outputs it is known as 9 to 2 multivalued ROM.

Thus it will perform with DNA sequences so it is called DNA multivalued ROM. In this ternary DNA computing, the fluorescent level is used to detect the DNA sequence. Fluorescence is defined as fluorescent molecules temporarily absorbing

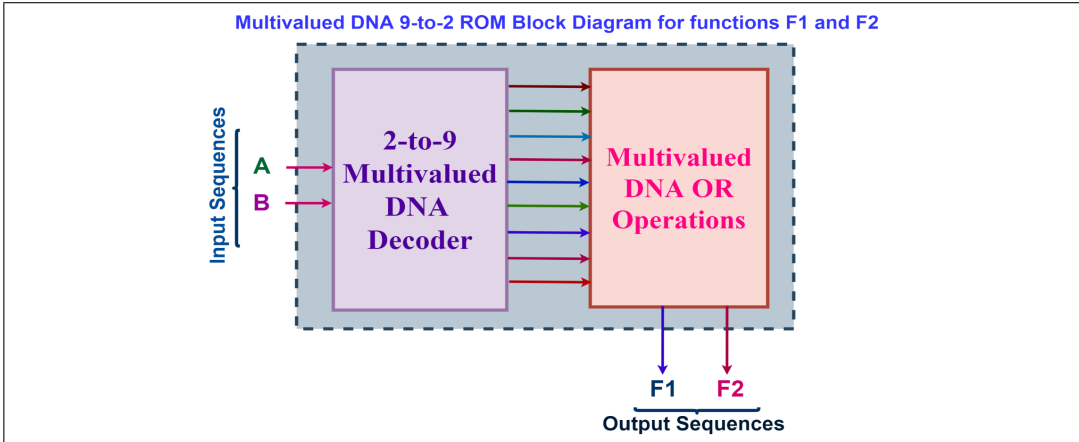


Figure 5: General Organization of multi-valued DNA 9-to-2 ROM .

electromagnetic wavelengths from the visible light spectrum and then emitting light at a lower energy level.

An architecture of a ROM is shown in Figure 3 and Figure 4 shows the circuit diagram of a ROM architecture. General Organization of multi-valued DNA 9-to-2 ROM is depicted in Figure 5.

Based on Figure 5, firstly DNA structure of multivalued Decoder with Block Diagram of multivalued DNA OR operations is illustrated in Figure 6. Secondly, using Figure 7 the Block Diagram of multivalued DNA Decoder with DNA structure of multivalued OR operations is depicted.

To perform multi-valued DNA ROM, we need a 2 to 9 decoder and minterms of decoder output as the OR gates input to produce desire multivalued ROM output functions F1 and F2. The design procedure of the proposed multi-valued DNA ROM is explained by the following steps:

Step 1:

First draw two input DNA sequence A and B. Three possible states for a DNA input sequence are the states ACCTAG, CAAGCT and TGGATC. These 3 will produce 9 combination of 2 input DNA sequence.

Step 2:

For each input, we need a ternary decoder for the selection of input values combinations. For example, if any input is A then three value can be performed as A0, A1 and A2. So the ternary decoder will select combinations from the values of inputs [30].

Step 3:

After the ternary decoder operations for input A and B, we will found A0, A1, A2

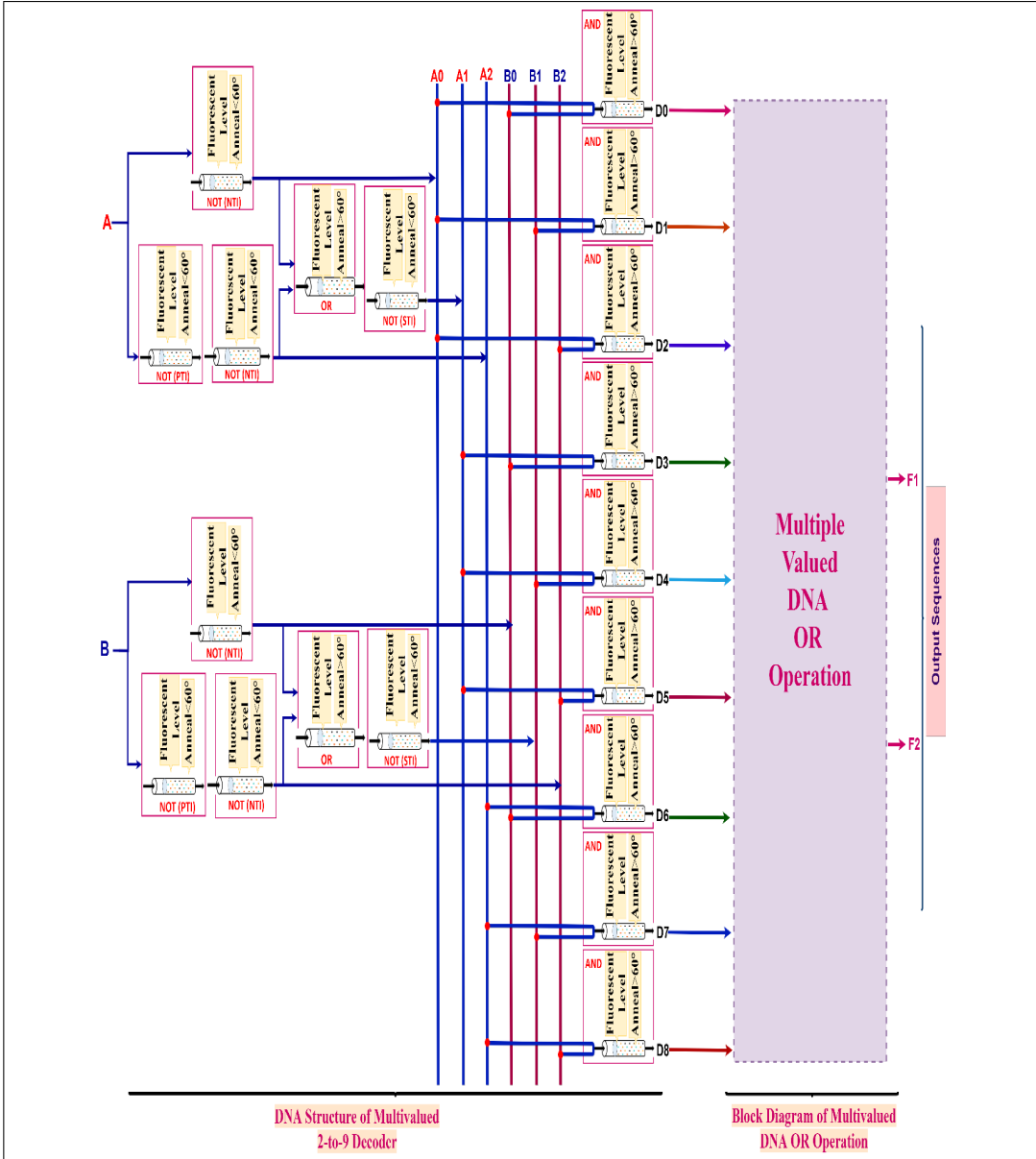


Figure 6: DNA structure of multivalued Decoder with Block Diagram of multivalued DNA OR operations.

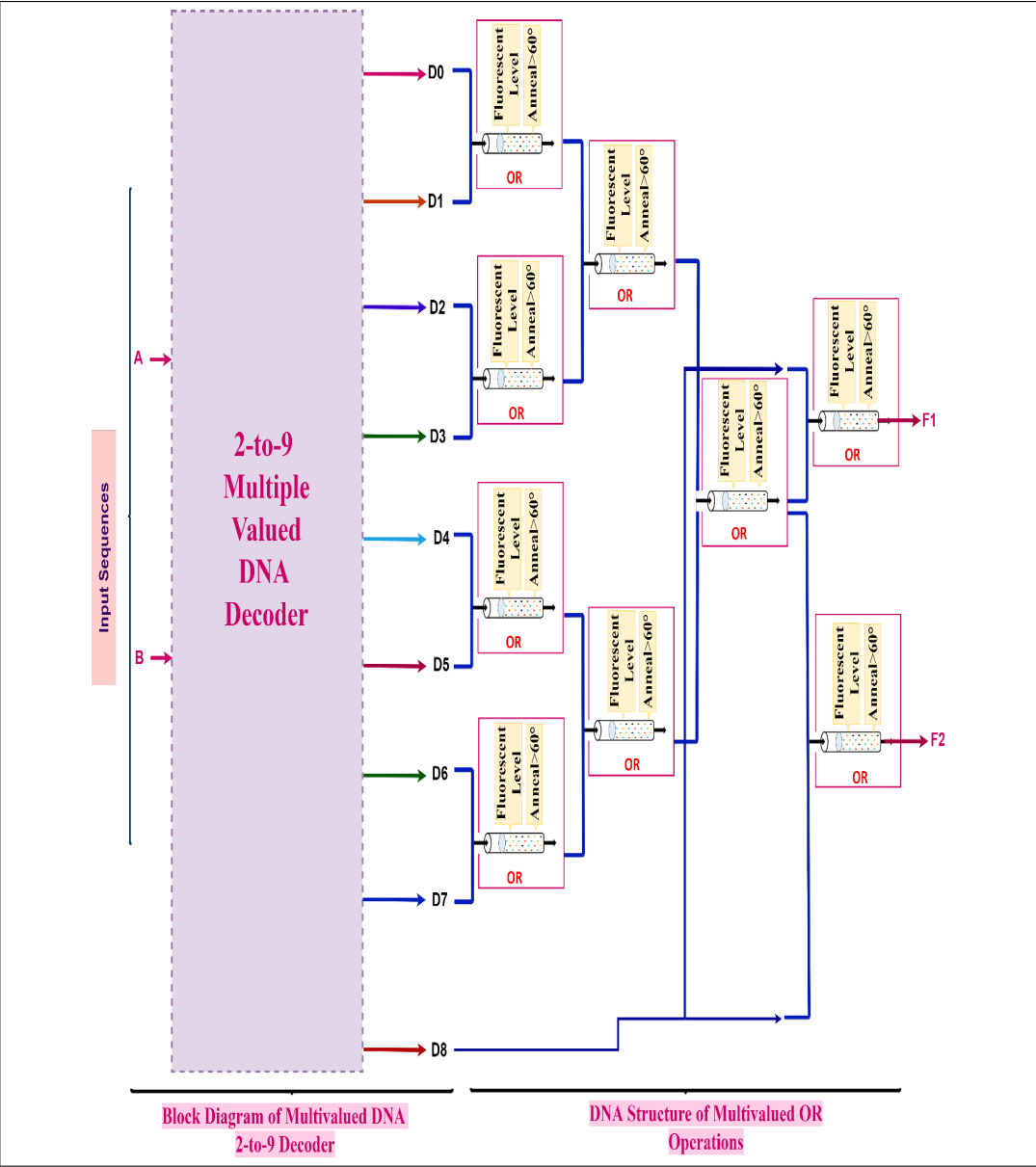


Figure 7: Block Diagram of multivalued DNA Decoder with DNA structure of multivalued OR operations.

and B0, B1, B2.

Step 4:

With the each input values of step 3, we need to perform AND operations that will produce 9 combination of 2 input sequences and enable any one output line. Step 5: From each combination of AND operations in Step 4, draw multivalued DNA OR gates with (D0, D1), (D2, D3), (D4, D5), (D6, D7).

Step 6:

From each outputs sequences of Step 5, again combine them with multivalued DNA OR gates, (D0, D1, D2, D3), (D4, D5, D6, D7).

Step 7:

From each outputs sequences of Step 6, again combine them with multivalued DNA OR gates as (D0, D1, D2, D3, D4, D5, D6, D7).

Step 8:

Finally, perform multivalued DNA OR operations with D8 and the output of step 7 to produce desired multivalued 9-to-2 DNA ROM for output functions F1 and F2. Truth Table of 9-to-2 DNA ROM is given in Table 4.

A	B	F1	F2
ACCTAG	ACCTAG	TGGATC	TGGATC
ACCTAG	CAAGCT	TGGATC	TGGATC
ACCTAG	TGGATC	TGGATC	TGGATC
CAAGCT	ACCTAG	TGGATC	TGGATC
CAAGCT	CAAGCT	TGGATC	TGGATC
CAAGCT	TGGATC	TGGATC	TGGATC
TGGATC	ACCTAG	TGGATC	TGGATC
TGGATC	CAAGCT	TGGATC	TGGATC
TGGATC	TGGATC	TGGATC	TGGATC

Table 4: Truth Table of 9-to-2 DNA ROM

The multivalued decoder operation is performed by nine multivalued AND gates

which are “MINIMUM OF” sequences respectively. So, with the two input DNA sequences and each of three states. From these combination, the corresponding multivalued AND operation outputs are the minimum of input sequence states. Ternary logic functions are those functions which have significance if a third value is acquainted with the binary logic. In this paper, ACCTAG, CAAGCT and TGGATC denote the ternary levels for basic logic gates to represent sequence ACCTAG is considered as “0”, sequence CAAGCT strands as “1” and TGGATC stands as “2”. For multivalued 2:9 decoder 9 outputs. D0, D1, D2, D3, D4, D5, D6, D7, D8 and two input A, B.

For input sequences A, B= ACCTAG, ACCTAG

- a. To perform value of input A0, A will go through the NOT (NTI), and produces TGGATC in the A0.
- b. To perform value of input A2, A will go through the NOT (PTI) and produce TGGATC then go through NOT (NTI) and produces ACCTAG in the A2.
- c. To perform value of input A1, the value of A0 and A2 go through DNA NOR (OR, NOT) Operations and produces ACCTAG in the A1.
- d. To perform value of input B0, B will go through the NOT (NTI), and produces TGGATC in the B0.
- e. To perform value of input B2, B will go through the NOT (PTI) and produce TGGATC then go through NOT (NTI) and produces ACCTAG in the B2.
- f. To perform value of input B1, the value of B0 and B2 go through DNA NOR (OR, NOT) Operations and produces ACCTAG in the B1.

$$\text{ACCTAG} = |0\rangle \text{ CAAGCT} = |1\rangle \text{ TGGATC} = |2\rangle$$

A= A0, A1, A2 = 2, 0, 0 and B= B0, B1, B2 = 2, 0, 0. With this input combinations, only input A0 and B0 is true. Thus the A0 and B0 are connected to output line D0 so only the D0 will TGGATC and rest of the gates will produce ACCTAG.

To perform 9 to 2 multivalued DNA ROM functions outputs,

1. For input sequences D0, D1 = TGGATC, ACCTAG, the multivalued OR gates will perform maximum operations of input sequences. So the output sequence of TGGATC.
2. For input sequences D2, D3 = ACCTAG, ACCTAG, the multivalued OR gates will perform maximum operations of input sequences. So the output sequence of ACCTAG.
3. For input sequences D4, D5 = ACCTAG, ACCTAG, the multivalued OR gates will perform maximum operations of input sequences. So the output sequence

of ACCTAG.

4. For input sequences D6, D7 = ACCTAG, ACCTAG the multivalued OR gates will perform maximum operations of input sequences. So the output sequence of ACCTAG.
5. Combine the output sequences TGGATC, ACCTAG from [i] and [ii] respectively as input sequences, the multivalued OR gates will perform maximum operations of input sequences. So the output sequence of TGGATC.
6. Combine the output sequences ACCTAG, ACCTAG from [iii] and [iv] respectively as input sequences, the multivalued OR gates will perform maximum operations of input sequences. So the output sequence of ACCTAG.
7. Combine the output sequences ACCTAG, ACCTAG from [iii] and [iv] respectively as input sequences, the multivalued OR gates will perform maximum operations of input sequences. So the output sequence of ACCTAG.
8. Combine the output sequences TGGATC, ACCTAG from [v] and [vi] respectively as input sequences, the multivalued OR gates will perform maximum operations of input sequences. So the output sequence of TGGATC.
9. Finally, Combine the output sequence TGGATC from [vii] and D8 (ACCTAG) as input sequences, the multivalued OR gates will perform maximum operations of input sequences. So the output sequence of TGGATC for the both functions output F1 and F2 multivalued 9 to 2 DNA ROM.

For input sequences A, B= ACCTAG, CAAGCT

A= A0, A1, A2 = 2, 0, 0 and B= B0, B1, B2= 0, 2, 0. With this input combinations, only input A0 and B1 is true. Thus the A0 and B1 are connected to output line D1 so only the D1 will TGGATC and rest of the gates will produce ACCTAG.

For Functions F1 and F2, perform OR operations among all decoder (D0-D8) outputs. So, $\max(D0, D1, D2, D3, D4, D5, D6, D7, D8) = \max(0, 2, 0, 0, 0, 0, 0, 0, 0) = 2$ will produce as the functions outputs F1 and F2 of multivalued 9 to 2 DNA ROM.

For input sequences A, B= ACCTAG, TGGATC

A= A0, A1, A2 = 2, 0, 0 and B= B0, B1, B2= 0, 0, 2. With this input combinations, only input A0 and B2 is true. Thus the A0 and B2 are connected to output line D2 so only the D2 will TGGATC and rest of the gates will produce ACCTAG.

For Functions F1 and F2, perform OR operations among all decoder (D0-D8) outputs. So, $\max(D0, D1, D2, D3, D4, D5, D6, D7, D8) = \max(0, 0, 2, 0, 0, 0, 0, 0, 0) = 2$ will produce as the functions outputs F1 and F2 of multivalued 9 to 2 DNA ROM.

For input sequences A, B= CAAGCT, ACCTAG

A= A0, A1, A2 = 0, 2, 0 and B= B0, B1, B2= 2, 0, 0. With this input combinations, only input A1 and B0 is true. Thus the A1 and B0 are connected to output line D3 so only the D3 will TGGATC and rest of the gates will produce ACCTAG.

For Functions F1 and F2, perform OR operations among all decoder (D0-D8) outputs. So, $\max (D0, D1, D2, D3, D4, D5, D6, D7, D8) = \max (0,0,0,2,0,0,0,0,0) = 2$ will produce as the functions outputs F1 and F2 of multivalued 9 to 2 DNA ROM.

For input sequences A, B= CAAGCT, CAAGCT

A= A0, A1, A2 = 0, 2, 0 and B= B0, B1, B2= 0, 2, 0. With this input combinations, only input A1 and B1 is true. Thus the A1 and B1 are connected to output line D4 so only the D4 will TGGATC and rest of the gates will produce ACCTAG.

For Functions F1 and F2, perform OR operations among all decoder (D0-D8) outputs. So, $\max (D0, D1, D2, D3, D4, D5, D6, D7, D8) = \max (0,0,0,0,2,0,0,0,0) = 2$ will produce as the functions outputs F1 and F2 of multivalued 9 to 2 DNA ROM.

For input sequences A, B= CAAGCT, TGGATC

A= A0, A1, A2 = 0, 2, 0 and B= B0, B1, B2= 0, 0, 2. With this input combinations, only input A1 and B2 is true. Thus the A1 and B2 are connected to output line D5 so only the D5 will TGGATC and rest of the gates will produce ACCTAG.

For Functions F1 and F2, perform OR operations among all decoder (D0-D8) outputs. So, $\max (D0, D1, D2, D3, D4, D5, D6, D7, D8) = \max (0,0,0,0,0,2,0,0,0) = 2$ will produce as the functions outputs F1 and F2 of multivalued 9 to 2 DNA ROM.

For input sequences A, B= TGGATC, ACCTAG

A= A0, A1, A2 = 0, 0, 2 and B= B0, B1, B2= 2, 0, 0. With this input combinations, only input A2 and B0 is true. Thus the A2 and B0 are connected to output line D6 so only the D6 will TGGATC and rest of the gates will produce ACCTAG.

For Functions F1 and F2, perform OR operations among all decoder (D0-D8) outputs. So, $\max (D0, D1, D2, D3, D4, D5, D6, D7, D8) = \max (0,0,0,0,0,0,2,0,0) = 2$ will produce as the functions outputs F1 and F2 of multivalued 9 to 2 DNA ROM.

For input sequences A, B= TGGATC, CAAGCT

A= A0, A1, A2 = 0, 0, 2 and B= B0, B1, B2= 0, 2, 0. With this input combinations, only input A2 and B1 is true. Thus the A2 and B1 are connected to output line D7 so only the D7 will TGGATC and rest of the gates will produce ACCTAG.

For Functions F1 and F2, perform OR operations among all decoder (D0-D8) outputs. So, $\max (D0, D1, D2, D3, D4, D5, D6, D7, D8) = \max (0,0,0,0,0,0,0,2,0) = 2$ will produce as the functions outputs F1 and F2 of multivalued 9 to 2 DNA ROM.

For input sequences A, B= TGGATC, TGGATC

A= A0, A1, A2 = 0, 0, 2 and B= B0, B1, B2= 0, 0, 2. With this input combinations, only input A2 and B2 is true. Thus the A2 and B2 are connected to output line D8 so only the D8 will TGGATC and rest of the gates will produce ACCTAG.

For Functions F1 and F2, perform OR operations among all decoder (D0-D8) outputs. So, $\max(D0, D1, D2, D3, D4, D5, D6, D7, D8) = \max(0,0,0,0,0,0,0,2) = 2$ will produce as the functions outputs F1 and F2 of multivalued 9 to 2 DNA ROM. Algorithm 2 represents the working procedure of DNA-based multi-valued ROM. Here, A and B are the inputs and the final outputs are F1 and F2. As A, B, F1, and F2 are multi-valued variables, ACCTAG or CAAGCT or TGGATC can be a value of individuals.

The temperature necessary to break the bonds between each pair of nucleotides is known as the melting point temperature of a sequence. Melting point temperatures rise in a nonlinear manner as the nucleotide sequence lengthens. The frequency of dinucleotides is used to calculate the melting point temperature. In this work we give and prove Theorems 1 to 3 based on [31].

Theorem 1. *The minimum melting temperature of a length 6 DNA sequence is 5 °C Celsius.*

Proof: The Melting of DNA sequences is an essential part of DNA computing. Using the modified method of Marmur Doty formula [31], the melting temperature (T_m) is calculated. This formula perfectly works for the short length oligonucleotides. The value of T_m is obtained using the following equation:

$$T_m = 2(A + T) + 4(C + G) - 7; \quad (11)$$

where

T_m = Melting temperature in °C;

A = Total Number of adenosine nucleotides in the given sequence;

T = Total Number of thymidine nucleotides in the given sequence;

C = Total Number of cytosine nucleotides in the given sequence;

G = Total Number of guanosine nucleotides in the given sequence; and

-7 = Correction factor In ACCTAG sequence, the number of A=2, C=2, T=1, and G=1.

So, $T_m = 2(3) + 4(3) - 7 = 11$

In TGGATC, the number of A=1, C=1, T=2, and G=2. Thus,

$T_m = 2(3) + 4(3) - 7 = 11$

When the combination of an oligonucleotide is length 6 sequence consisting of only C and G nucleotides, the (T_m) will be $T_m = 2(0) + 4(6) - 7 = 17$

On the other hand, when the combination of an oligonucleotide is length 6 sequence consisting of only A and T nucleotides, the (T_m) will be $T_m = 2(6) + 4(0) - 7 = 5$ So it is clear that the minimum melting temperature of a length 6 DNA sequence is

Algorithm 2 Multi-Valued DNA-Based ROM

Input: A, B

Output: F1, F2;

The value of A, B, F1, and F2 can be ACCTAG or CAAGCT or TGGATC

Begin

while $i \leq n$ **do** $P = DO_DNA_Decoder(A_i, B_i);$ \triangleright Decoder generates D0-D8 $P0 \leftarrow DO_DNA_Multiple_OR(D0, D1);$ $P1 \leftarrow DO_DNA_Multiple_OR(D2, D3);$ $P2 \leftarrow DO_DNA_Multiple_OR(D4, D5);$ $P3 \leftarrow DO_DNA_Multiple_OR(D6, D7);$ $P4 \leftarrow DO_DNA_Multiple_OR(P0, P1);$ $P5 \leftarrow DO_DNA_Multiple_OR(P2, P3);$ $P6 \leftarrow DO_DNA_Multiple_OR(P4, P5);$ $F1, F2 \leftarrow DO_DNA_Multiple_OR(P6, D8);$ **end while****procedure** $DO_DNA_Multiple_OR(X, Y)$ **if** X equals Y **then** $Do_Result(X);$ **else if** $X < Y$ **then** $Do_Result(Y);$ **else if** $X > Y$ **then** $Do_Result(X);$ **end if****end procedure****procedure** $Do_Result(Q)$ $P \leftarrow fluorescent_get_value(Q);$ **if** P equals 0 **then****return** ACCTAG**else if** P equals 1 **then****return** CAAGCT**else if** P equals 2 **then****return** TGGATC**end if****end procedure**End

5 °C Celsius.

Theorem 2. *If a short-length DNA sequence consists of only A and T nucleotides, then the required melting temperature is $T_m=2x-7$, where x is the total number of nucleotides.*

Proof: The modified method of Marmur Doty formula [31] helps to calculate the melting temperature of a DNA sequence. The melting temperature of calculation equation is given as follows:

$$T_m = 2(A + T) + 4(C + G) - 7 \text{ Here,}$$

T_m = Melting temperature in °C;

A, T, C, and G are the number of adenosine, thymidine, cytosine and guanosine nucleotides in the given sequence, sequentially.

-7 = Correction factor

Let $A+T=x$ and $C+G=y$.

Then Equation (10) can be expressed as follows:

$T_m = 2x + 4(y) - 7$ (11) When a sequence consists of only A and T nucleotides, the value of $y=0$. In that case, Equation (11) will be

$$T_m = 2x + 4(0) - 7 = 2x - 7$$

So, it is clear that if a short-length DNA sequence consists of only A and T nucleotides, then the required melting temperature is $T_m=2x-7$, where x is the total number of nucleotides.

Theorem 3. *If a short-length DNA sequence consists of only C and G nucleotides, then the required melting temperature is $T_m=4y-7$, where y is the total number of nucleotides.*

Proof: The modified method of Marmur Doty formula [31] helps to calculate the melting temperature of a DNA sequence. The equation of melting temperature is given below.

$$T_m = 2(A + T) + 4(C + G) - 7;$$

where T_m = Melting temperature in °C; and

A, T, C and G are the number of adenosine, thymidine, cytosine and guanosine nucleotides in the given sequence, sequentially, where

-7 = Correction factor.

Let $A+T=x$ and $C+G=y$.

Then Equation (10) can be expressed as follows:

$$T_m = 2(x) + 4(y) - 7 \text{ (12)}$$

When a sequence consists of only A and T nucleotides, the value of $y=0$. In that case, Equation (12) will be

$$Tm = 2(0) + 4(y) - 7 = 4y - 7$$

So, it can say that if a short-length DNA sequence consists of only C and G nucleotides, then the required melting temperature is $Tm=4y-7$, where y is the total number of nucleotides.

Table 7 shows the melting temperature of different length's DNA sequences consisting of A and T. Table 6 and Table 7 show the melting temperature of different length's DNA sequences consisting of "C and G" and "A,T, C, and G", respectively. For the sequences consisting of C and G require the maximum melting temperature and the sequences consisting of A and T require the minimum melting temperature.

4 Conclusions

Security experts have identified a bug in Intel's read-only memory that they believe is unfixable, leaving all Intel devices. The fact is that once an attacker has a static circuit, it is just a matter of time before they can reverse engineer with its configuration. As a result, dynamic ROM (Read Only Memory) configurations are required, and biological multi-valued methods have been invented to imitate existing vulnerable static technologies in order to ensure reliable storage. Multi-valued DNA (Deoxyribose Nucleic Acid) computing introduces an approach for generating information that can be stored and retrieved reliably within such a DNA sequence. In this paper, a multi-valued DNA ROM architecture is designed with the help of proposed multi-valued DNA operations algorithms. This multi-valued DNA ROM has the capability of long-life storage, low-cost fabrication, and higher memory density. Researchers at UC Davis, the University of Washington and Emory University have developed a memory technology that applies DNA bases to encode information directly. The researchers have demonstrated the capability to create DNA-based ROM that is programmable and can interface seamlessly with current electronic devices. The technology applies the self-assembly and electrical conductance properties of DNA to create crosswire (X-wire) nanostructures that simulate the "ones and zeroes" that currently form the basis for electronic storage of digital information [13], [32].

References

- [1] F. M. Raymo, Digital processing and communication with molecular switches, *Advanced Materials* 14 (6) (2002) 401–414.
- [2] G. Dilek, E. U. Akkaya, Novel squaraine signalling zn (ii) ions: three-state fluorescence response to a single input, *Tetrahedron Letters* 41 (19) (2000) 3721–3724.

Length	DNA Sequences	Heat (°C)
2	AT	-3
3	ATA	-1
4	TAAT	1
5	TTAAT	3
6	ATATAT	5
7	ATAATTA	7
8	ATATATA	9
9	TTAATAAAT	11
10	TTAATATAT	13
11	AATTATAATAA	15
12	TAATATAATAAT	17
13	TATTATAATATAT	19
14	ATAATTATAATTAT	21

Table 5: Melting Temperature of DNA Sequences (Sequences consisting of A and T)

- [3] R. Merindol, A. Walther, Materials learning from life: concepts for active, adaptive and autonomous molecular systems, *Chemical Society Reviews* 46 (18) (2017) 5588–5619.
- [4] R. L. Adams, *The biochemistry of the nucleic acids*, Springer Science & Business Media, 2012.
- [5] L. M. Adleman, Molecular computation of solutions to combinatorial problems, *Science* 266 (5187) (1994) 1021–1024.
- [6] G. Seelig, D. Soloveichik, D. Y. Zhang, E. Winfree, Enzyme-free nucleic acid logic circuits, *science* 314 (5805) (2006) 1585–1588.
- [7] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, E. Shapiro, Programmable and autonomous computing machine made of biomolecules, *Nature* 414 (6862) (2001) 430–434.

Length	DNA Sequences	Heat (°C)
2	GC	1
3	CGC	5
4	GCCG	9
5	CGGCG	13
6	GCCGGC	17
7	CGCCGGC	21
8	GCCGCGCG	25
9	CGCCGGCGC	29
10	GCGCGGCGCG	33
11	CGCGCGGCGCC	37
12	GCGCGCGCGCGC	41
13	GCCGGCCGCGCGC	45
14	CGCCGGCCGCGCGC	49

Table 6: Melting Temperature of DNA Sequences (Sequences consisting of C and G))

- [8] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, E. Shapiro, An autonomous molecular computer for logical control of gene expression, *Nature* 429 (6990) (2004) 423–429.
- [9] M. N. Stojanovic, D. Stefanovic, A deoxyribozyme-based molecular automaton, *Nature biotechnology* 21 (9) (2003) 1069–1074.
- [10] L. Qian, E. Winfree, Scaling up digital circuit computation with dna strand displacement cascades, *Science* 332 (6034) (2011) 1196–1201.
- [11] L. Qian, E. Winfree, J. Bruck, Neural network computation with dna strand displacement cascades, *Nature* 475 (7356) (2011) 368–372.
- [12] K. M. Cherry, L. Qian, Scaling up molecular pattern recognition with dna-based winner-take-all neural networks, *Nature* 559 (7714) (2018) 370–376.
- [13] H. M. H. Babu, Reversible and dna computing.

Length	DNA Sequences	Heat (°C)
2	AC	-1
3	CTG	3
4	TCAG	5
5	GACTA	7
6	TCACAG	11
7	CTGATCA	13
8	GATCTACT	15
9	TCACAGGCT	21
10	CTGCTGATCA	23
11	ACGATCTACTT	25
12	GATCTACTTCAG	27
13	TACTTCAGATCTA	31
14	ACGATCTACTTCAG	33

Table 7: Melting Temperature of DNA Sequences (Sequences consisting of A, T, C, and G)

- [14] R. Deaton, R. C. Murphy, J. Rose, M. Garzon, D. R. Franceschetti, S. Stevens, A dna based implementation of an evolutionary search for good encodings for dna computation, in: Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97), IEEE, 1997, pp. 267–271.
- [15] C. Zhang, L. Ge, Y. Zhuang, Z. Shen, Z. Zhong, Z. Zhang, X. You, Dna computing for combinational logic, Science China Information Sciences 62 (6) (2019) 1–16.
- [16] A. Extance, How dna could store all the world's data, Nature News 537 (7618) (2016) 22.
- [17] J. D. Watson, F. Crick, Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid, American Journal of Psychiatry 160 (4) (2003) 623–624.

- [18] H. Echols, M. F. Goodman, Fidelity mechanisms in dna replication, *Annual review of biochemistry* 60 (1) (1991) 477–511.
- [19] V. Gupta, S. Parthasarathy, M. Zaki, Arithmetic and logic operations with dna.
- [20] H. Lederman, J. Macdonald, D. Stefanovic, M. N. Stojanovic, Deoxyribozyme-based three-input logic gates and construction of a molecular full adder, *Biochemistry* 45 (4) (2006) 1194–1199.
- [21] E. Pérez-Inestrosa, J.-M. Montenegro, D. Collado, R. Suau, J. Casado, Molecules with multiple light-emissive electronic excited states as a strategy toward molecular reversible logic gates, *The Journal of Physical Chemistry C* 111 (18) (2007) 6904–6909.
- [22] H. M. H. Babu, T. Sasao, Design of multiple-output networks using time domain multiplexing and shared multi-terminal multiple-valued decision diagrams, in: *Proceedings. 1998 28th IEEE International Symposium on Multiple-Valued Logic* (Cat. No. 98CB36138), IEEE, 1998, pp. 45–51.
- [23] N. J. Lisa, H. M. H. Babu, Design of a compact ternary parallel adder/subtractor circuit in quantum computing, in: *2015 IEEE International Symposium on Multiple-Valued Logic*, IEEE, 2015, pp. 36–41.
- [24] H. H. Babu, T. Sasao, Representations of multiple-output switching functions using multiple-valued pseudo-kronecker decision diagrams, in: *Proceedings 30th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2000)*, IEEE, 2000, pp. 147–152.
- [25] S. Lin, Y.-B. Kim, F. Lombardi, Cntfet-based design of ternary logic gates and arithmetic circuits, *IEEE transactions on nanotechnology* 10 (2) (2009) 217–225.
- [26] A. Heung, H. Mouftah, Depletion/enhancement cmos for a lower power family of three-valued logic circuits, *IEEE Journal of Solid-State Circuits* 20 (2) (1985) 609–616.
- [27] L. J. Kricka, P. Fortina, Analytical ancestry:“firsts” in fluorescent labeling of nucleosides, nucleotides, and nucleic acids, *Clinical Chemistry* 55 (4) (2009) 670–683.
- [28] A. Dhande, V. Ingole, Design and implementation of 2 bit ternary alu slice, in: *Proc. Int. Conf. IEEE-Sci. Electron., Technol. Inf. Telecommun*, Vol. 17, 2005.
- [29] M. H. Khan, Design of reversible/quantum ternary multiplexer and demultiplexer., *Engineering letters* 13 (3).
- [30] R. Jaber, A. Elhajj, L. Nimri, A. Haidar, A novel implementation of ternary decoder using cmos dpl binary gates, in: *2018 International Arab Conference on Information Technology (ACIT)*, IEEE, 2018, pp. 1–3.
- [31] J. Marmur, P. Doty, Determination of the base composition of deoxyribonucleic acid from its thermal denaturation temperature, *Journal of molecular biology* 5 (1) (1962) 109–118.
- [32] J. L. Hihath, Dna-based read-only memory (rom) for data storage applications, Office of Research, USA.

TERNARY FUNCTIONS WITH BENT REED-MULLER-FOURIER SPECTRA

CLAUDIO MORAGA

Technical University of Dortmund, 44221 Dortmund, Germany

Technical University "Federico Santa María", Valparaíso, Chile

`claudio.moraga@tu-dortmund.de`

RADOMIR STANKOVIĆ

*Department of Computer Science, Faculty of Electronic Engineering, 18000 Niš,
Serbia*

Mathematical Institute of SAsA, 11000 Belgrade, Serbia

`radomir.stankovic@gmail.com`

MILENA STANKOVIĆ

University of Niš, Faculty of Electronic Engineering, 18 000 Niš, Serbia

`milstankovic@gmail.com`

Abstract

Ternary functions which have a bent Reed-Muller-Fourier (RMF) spectrum are introduced. Necessary conditions for a function to have this property are established. A classification of ternary 2-place functions having a bent RMF spectrum in 6 classes is given and it is shown that these classes are related by spectral invariance operations. Moreover, a compact database of those ternary functions with bent RMF spectra is provided. Ternary bent functions having the same value vector as their respective RMF bent spectra are known as fixed points. The paper generalizes this concept to rotational fixed points when the value vector of a function equals its spectra shifted by a constant. Finally, a method is introduced to generate n -place ternary functions with bent RMF spectrum when $n > 2$.

Keywords: RMF transform, bent functions, restricted tensor sum.

1 Introduction

A preliminary step towards the Reed-Muller-Fourier transform, (short RMF transform) was introduced in [18], where the convolution product of Gibbs [2] was extended to the multiple-valued case. The RMF transform was introduced then in 1993 [19] for the multiple-valued domain, preserving important features of the binary Reed-Muller transform, like the Kronecker product structure as well as being self-inverse, and the lower triangular structure of the Discrete Fourier transform. The RMF transform generates a bijection in the set of p -valued functions: the RMF spectrum of a p -valued function is also a p -valued function. Particular properties of the RMF transform have been studied in [9] – [12], [20] – [24] and [27] – [29].

Bent functions were introduced in 1976 [16], as the most non-linear binary functions. This attracted the interest of people working in Coding Theory, and in Cryptography. See e.g. [1]. The extension of bent functions to the multiple-valued domain was introduced almost a decade later [5]. Much work has been done on multiple-valued bent function considered as challenging mathematical objects, see e.g. [6], [8] and Chapter 15 of [25].

2 Formalisms

The following notation will be used in this paper.

Let $f : (Z_p)^n \rightarrow Z_p$ denote a p -valued function. \mathbf{F} will denote its value vector. If no confusion arises, \mathbf{F} may also be called *function*. If $p = 3$, then f will be called ternary. Let \mathbf{T} denote the RMF transform matrix and let \mathbf{R}_f denote the RMF spectrum of f . Similarly, \mathbf{S}_f will denote the Vilenkin-Chrestenson spectrum of the complex encoding of f as defined below. Furthermore, if \mathbf{Q} denotes a ternary vector, then $\mathbf{1}^Q$ will denote a column vector of the same length as \mathbf{Q} , with all entries equal to 1. Similarly for $\mathbf{0}^Q$ and $\mathbf{2}^Q$. If the length of \mathbf{Q} should be explicitly given, a numerical exponent will be used. Moreover, let $\mathbf{1}^q$ denote a row tuple of 1 's of *length* = ((length of \mathbf{Q}) – 1) and similarly for $\mathbf{0}^q$. Finally, if \mathbf{A} is a matrix, then $\text{vec}(\mathbf{A})$ concatenates the columns of \mathbf{A} to build a vector [3].

2.1 The RMF transform

Definition 1. [18]

The Gibbs convolution product (\times) of multiple-valued functions when $n = 1$ is given as follows:

$$\begin{bmatrix} 2 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 2 & 2 \end{bmatrix}, \quad \begin{bmatrix} 3 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 3 & 2 & 3 & 0 \\ 3 & 3 & 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 & 0 \\ 4 & 2 & 4 & 0 & 0 \\ 4 & 3 & 2 & 1 & 0 \\ 4 & 4 & 4 & 4 & 4 \end{bmatrix}, \\
 \begin{bmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 2 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 3 & 6 & 1 & 0 & 0 & 0 & 0 & 0 \\ 8 & 4 & 3 & 4 & 8 & 0 & 0 & 0 & 0 \\ 8 & 5 & 8 & 1 & 4 & 1 & 0 & 0 & 0 \\ 8 & 6 & 3 & 2 & 3 & 6 & 8 & 0 & 0 \\ 8 & 7 & 6 & 8 & 1 & 3 & 2 & 1 & 0 \\ 8 & 8 & 8 & 2 & 2 & 2 & 8 & 8 & 8 \end{bmatrix}.$$

 Figure 1: Matrices $\mathbf{X}_{3RMF(1)}$, $\mathbf{X}_{4RMF(1)}$, $\mathbf{X}_{5RMF(1)}$, and $\mathbf{X}_{9RMF(1)}$.

Let $f, g : Z_p \rightarrow Z_p$ such that $(f \times g)(0) = 0$, and for $x > 0$:

$$(f \times g)(x) = \sum_{s=0}^{x-1} f(x-1-s) \cdot g(s) \pmod{p}.$$

Definition 2. [19]

The fundamental basis for the RMF transform is the following: $[x^{*0} x^{*1} \dots x^{*(p-1)}]$, where x^{*0} is defined to be the constant $p-1$ for all x , and for $1 \leq j \leq p-1$, the powers x^{*j} are calculated as the j th-fold Gibbs product of x^{*0} with itself.

Preserving the original notation [19], the matrix version of the basis in Definition 2 is called $\mathbf{X}_{pRMF(1)}$. Fig. 1 shows the matrices for $p = 3, 4, 5$ and 9.

It is simple to show that the matrices are self-inverse in the ring $(Z_p, +, \cdot)$.

From Fig. 1 it may be noticed that the matrices are lower triangular and, additionally, if p is a prime, that $\mathbf{X}_{pRMF(1)}$ is skew-symmetric (i.e. symmetric with respect to the diagonal with positive slope). Notice that if $p = 9$ the bottom row of the matrix is $8 \ 8 \ 8 \ 2 \ 2 \ 2 \ 8 \ 8 \ 8$, i.e. it is not enough that p is odd to achieve skew-symmetry.

Definition 3. Let $\mathbf{T}(n)$ denote the RMF transform matrix compatible with functions in n variables and preserving the distribution of values indicated in Definition 2. Then:

$$\mathbf{T}(n) = (p-1)^{n+1}(\mathbf{X}_{pRMF(1)})^{\otimes n} \bmod p \quad (1)$$

where the exponent $\otimes n$ indicates the n -fold Kronecker product of $\mathbf{X}_{pRMF(1)}$ with itself and $(p-1)^{n+1}$ is a normalizing factor. If n is odd, it is simple to realize that the matrix $\mathbf{X}_{pRMF(1)} \bmod p$ is lower triangular and that its first column is a column with all entries equal to $p-1$ as in Definition 2. Furthermore, the normalizing factor becomes $(p-1)^{even} \equiv 1 \bmod p$. If n is even, the matrix $\mathbf{X}_{pRMF(1)} \bmod p$ is lower triangular and its first column is a column with all entries equal to $(p-1)^n \equiv 1 \bmod p$. However, the normalizing factor becomes $(p-1)^{odd} \equiv (p-1) \bmod p$. Therefore, the transform matrix follows the values distribution as in Definition 2.

Recall that $\mathbf{X}_{pRMF(1)}$ is self-inverse. From the fact that the inverse of the Kronecker product of matrices equals the Kronecker product of the inverse of the matrices [3], with (1) follows that $\mathbf{T}(n)$ is also self-inverse.

Example 1. Let $p = 3$. Then, $\mathbf{X}_{3RMF(1)} \bmod 3 = \begin{bmatrix} 2 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 2 & 2 \end{bmatrix}$. With (1):

$$\begin{aligned} \mathbf{T}(2) &= 2^3 \cdot \begin{bmatrix} 2 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 2 & 2 \end{bmatrix}^{\otimes 2} \bmod 3 = 2 \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 \\ 2 & 1 & 0 & 2 & 1 & 0 & 2 & 1 & 0 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}. \end{aligned}$$

The matrix obtained for $\mathbf{T}(2)$ is conform with Definition 2.

Notice that the above is not just a particular property of the ternary transform. For any p , $\mathbf{X}_{pRMF(1)}$ starts with a column whose entries have the value $p-1$. Recall that for any n , $\mathbf{T}(n) = (p-1)^{n+1}(\mathbf{X}_{pRMF(1)})^{\otimes n} \bmod p$. It is simple to see that the n -fold Kronecker product of $\mathbf{X}_{pRMF(1)}$ with itself will produce a lower triangular matrix with a first column with entries equal to $(p-1)^n$. Since the normalizing factor is $(p-1)^{n+1}$, all entries of the first column will become $(p-1)^{2n+1} = (p-1)^{odd} \equiv (p-1) \bmod p$. The transform matrix becomes conform with Definition 2.

Lemma 1. For any positive integers m and n , $\mathbf{T}(m+n) = (p-1)(\mathbf{T}(m) \otimes \mathbf{T}(n)) \bmod p$.

Proof: With (1),

$$\begin{aligned} (p-1)(\mathbf{T}(m) \otimes \mathbf{T}(n)) &= (p-1) \left[\langle (p-1)^{m+1}(\mathbf{X}_{pRMF(1)})^{\otimes m} \rangle \right. \\ &\quad \left. \otimes \langle (p-1)^{n+1}(\mathbf{X}_{pRMF(1)})^{\otimes n} \rangle \right] \bmod p \\ &= (p-1)^{m+n+3} \langle (\mathbf{X}_{pRMF(1)})^{\otimes m} \rangle \otimes \langle (\mathbf{X}_{pRMF(1)})^{\otimes n} \rangle \\ &= (p-1)^{m+n+1} (\mathbf{X}_{pRMF(1)})^{\otimes m+n} \bmod p. \end{aligned}$$

The assertion follows, since the last expression represents $\mathbf{T}(m+n)$ according to (1).

The RMF transform has found applications e.g. in Signal Processing [21], [23] and in Pattern Analysis [12]. Moreover, it has been shown [7] that the RMF transform applied to the value vector of a ternary function, preserves any permutation of the arguments of the function. The RMF transform applied to Rotational-Symmetric ternary functions was studied in [13]. Other properties of the RMF transform have been studied in [27], [28], [29].

2.2 Bent Functions

From the mathematical point of view, some of the main tasks related to ternary bent functions are: generation, characterization, classification, and counting. Generation of ternary bent functions has been considered in e.g. [8], [15], [22], [24], [25], and classification of ternary bent functions in e.g. [14], [24]. The number of ternary bent functions for $n = 1$, (18), and for $n = 2$, (486), has been given in [8]. These "small" numbers have allowed to give proof of some properties simply by exhaustive search. Large numbers of ternary bent functions for some classes when $n = 3$ have been reported in [17]. Exhaustive search starts reaching its practical limits. Recall that the total number of ternary functions when $n = 3$ is $3^{27} \approx 7.6 \cdot 10^{12}$.

Definition 4. Let $\xi = \exp(2\pi i/3)$, where $i = \sqrt{-1}$, be a primitive cubic root of unity and let $h : \{0, 1, 2\} \rightarrow \{1, \xi, \xi^2\}$, where $\xi^2 = \xi^*$, the complex conjugate of ξ . The function h is known as the complex encoding function.

Definition 5. The Vilenkin-Chrestenson functions [4], [26] in matrix form are given by columns of the matrix

$$\mathbf{C}(n) = (\mathbf{C}(1))^{\otimes n} \quad \text{where} \quad \mathbf{C}(1) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \xi & \xi^2 \\ 1 & \xi^2 & \xi \end{bmatrix}. \quad (2)$$

The exponent $\otimes n$ indicates the n -fold Kronecker product of $\mathbf{C}(1)$ with itself. The Vilenkin-Chrestenson transform matrix is defined as the complex-conjugate of $\mathbf{C}(n)$.

With respect to characterization of ternary bent functions, it is known [8], [25] that if an n -place ternary function is bent, then the Vilenkin-Chrestenson spectrum of its complex encoding is "flat", meaning that the absolute value of all its spectral coefficients equals $3^{n/2}$.

Definition 6. Given a ternary function f in n variables, its Vilenkin-Chrestenson spectrum \mathbf{S}_f is calculated as follows:

$$\mathbf{S}_f = \mathbf{C}^*(n) \cdot h(\mathbf{F}), \quad (3)$$

where the complex encoding function h is applied to each of the elements of the value vector \mathbf{F} .

3 Bent RMF spectra of ternary functions

In [10], ternary functions with a bent Reed-Muller spectrum were studied. The present paper follows a similar line, but now related to Reed-Muller-Fourier spectra. Since the RM transform for ternary functions is not self-inverse, sequences of applications of the RM-transform could be applied until recovering the initial function. These "cycles" could have different lengths and meaning. Since the RMF transform is self-inverse, cycles become trivial and have length of 2. Therefore, the focus of the present study is oriented to the classification and characterization of ternary functions which exhibit a bent RMF spectrum.

Definition 7. Given a ternary function f in n variables, with value vector \mathbf{F} , its RMF spectrum is calculated as follows:

$$\mathbf{R}_f = \mathbf{T}(n) \cdot \mathbf{F} \bmod 3. \quad (4)$$

Lemma 2. *A necessary condition for a ternary 1-place function to have a bent RMF spectrum is that $f(2) \neq 0$.*

Proof:

$$\mathbf{T}(1) \cdot \mathbf{F} = \begin{bmatrix} 2 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 2 & 2 \end{bmatrix} \cdot \begin{bmatrix} f(0) \\ f(1) \\ f(2) \end{bmatrix} = \begin{bmatrix} 2f(0) \\ 2f(0) + f(1) \\ 2(f(0) + f(1) + f(2)) \end{bmatrix}.$$

The complex encoding of $\begin{bmatrix} 2f(0) \\ 2f(0) + f(1) \\ 2(f(0) + f(1) + f(2)) \end{bmatrix}$ is

$$\begin{bmatrix} \xi^{2f(0)} \\ \xi^{2f(0)+f(1)} \\ \xi^{2(f(0)+f(1)+f(2))} \end{bmatrix} = \xi^{2f(0)} \begin{bmatrix} 1 \\ \xi^{f(1)} \\ \xi^{2(f(1)+f(2))} \end{bmatrix}. \quad (5)$$

The coefficient $S(0)$ of its Vilenkin-Chrestenson spectrum is computed as $\xi^{2f(0)}(1 + \xi^{f(1)} + \xi^{2(f(1)+f(2))})$.

i) If $f(1) = 0$, $|S(0)| = |2 + \xi^{2f(2)}|$. If $f(2) = 0$, then $|S(0)| = 3$, however, should the spectrum be bent, then $|S(0)|$ should equal $\sqrt{3}$.

If $f(2) \neq 0$, then $\xi^{f(2)} = \xi$ or ξ^2 ; $|S(0)| = |2 + \xi| = |2 + \xi^2| = \sqrt{3}$.

ii) If $f(1) \neq 0$ and $f(2) = 0$, then $(1 + \xi^{f(1)} + \xi^{2(f(1)+f(2))}) = (1 + \xi^{f(1)} + \xi^{2f(1)}) = 0$, and therefore, $|S(0)| = 0$.

iii) If $f(1) = f(2) \neq 0$, then $(1 + \xi^{f(1)} + \xi^{2(f(1)+f(2))}) = (1 + 2\xi^{f(1)})$ and $|(1 + 2\xi^{f(1)})| = \sqrt{3}$.

If $f(1) = 2f(2) \neq 0$, then $(1 + \xi^{f(1)} + \xi^{2(f(1)+f(2))}) = (2 + \xi^{f(1)})$ and $|(2 + \xi^{f(1)})| = \sqrt{3}$.

To be flat, the absolute value of all spectral coefficients should be $\sqrt{3}$. Therefore $f(2)$ should not be 0. \square

Lemma 3. *The RMF transform preserves the sum mod 3 of two vectors (of the same length), the scaling by 2 of a function as well as (up to normalization) the Kronecker product of two vectors of not necessarily the same length. The RMF transform does not preserve the tensor sum \boxplus of two vectors. (The tensor sum [8] is also known as Kronecker sum [3]).*

Proof:

i) Let f and g be ternary functions in n variables. The following holds:

$$\mathbf{T}(n) \cdot (\mathbf{F} \oplus \mathbf{G}) = (\mathbf{T}(n) \cdot \mathbf{F}) \oplus (\mathbf{T}(n) \cdot \mathbf{G}) = \mathbf{R}_f \oplus \mathbf{R}_g.$$

ii) Let $\mathbf{G} = 2 \cdot \mathbf{F}$. Then $\mathbf{T}(n) \cdot \mathbf{G} = \mathbf{T}(n) \cdot 2 \cdot \mathbf{F} = 2 \cdot (\mathbf{T}(n) \cdot \mathbf{F})$. It follows that $\mathbf{R}_g = 2\mathbf{R}_f$.

iii) Let f and g be ternary functions in m and n variables, respectively. From Lemma 1, the following holds:

$$\begin{aligned} \mathbf{T}(m+n) \cdot (\mathbf{F} \otimes \mathbf{G}) &= (p-1)(\mathbf{T}(m) \otimes \mathbf{T}(n)) \cdot (\mathbf{F} \otimes \mathbf{G}) \\ &= (p-1)[(\mathbf{T}(m) \cdot \mathbf{F}) \otimes (\mathbf{T}(n) \cdot \mathbf{G})] \\ &= (p-1)(\mathbf{R}_f \otimes \mathbf{R}_g). \end{aligned}$$

iv) Recall that [8]

$$(\mathbf{F} \boxplus \mathbf{G}) = (\mathbf{F} \otimes \mathbf{1}^G) \oplus (\mathbf{1}^F \otimes \mathbf{G}). \quad (6)$$

Then: ¹

$$\begin{aligned} \mathbf{T}(m+n) \cdot (\mathbf{F} \boxplus \mathbf{G}) &= (p-1)(\mathbf{T}(m) \otimes \mathbf{T}(n)) \cdot ((\mathbf{F} \otimes \mathbf{1}^G) \oplus (\mathbf{1}^F \otimes \mathbf{G})) \\ &= (p-1)(\mathbf{T}(m) \otimes \mathbf{T}(n)) \cdot (\mathbf{F} \otimes \mathbf{1}^G) \\ &\quad \oplus (p-1)(\mathbf{T}(m) \otimes \mathbf{T}(n)) \cdot (\mathbf{1}^F \otimes \mathbf{G}) \quad (7) \\ &= (p-1)[\mathbf{T}(m) \cdot \mathbf{F} \otimes \mathbf{T}(n) \cdot (\mathbf{1}^G) \\ &\quad \oplus \mathbf{T}(m) \cdot (\mathbf{1}^F) \otimes \mathbf{T}(n) \cdot \mathbf{G}] \\ &= (p-1)[\mathbf{R}_f \otimes [1, \mathbf{0}^g]^T \oplus [1, \mathbf{0}^f]^T \otimes \mathbf{R}_g]. \end{aligned}$$

It is simple to see that Eqs. 6 and (7) have a different structure. Eq. (7) does not represent $\mathbf{R}_f \boxplus \mathbf{R}_g$. The RMF transform preserves the Kronecker product, but it does not preserve the tensor sum of ternary functions. \square

Lemma 4. *There are 18 1-place ternary functions, which have a bent RMF spectrum. 12 of them are bent, 2 are fixed points and 6 are rotational fixed points.*

Proof: Exhaustive search. \square

Table 1 shows the value vectors of 9 1-place ternary functions and their bent RMF spectra. Scaling by 2 gives the other 9 functions, since scaling by 2 is a spectral invariance operation and preserves bentness [17]. Moreover, as shown in Lemma 2, the RMF transform preserves a scaling by 2.

¹To make the representation of equations more compact here and mostly in what follows, elements of vectors will be separated by commas instead of spaces.

Ternary 1-place function	Bent RMF spectra $n = 1$	Function class
$[0\ 0\ 2]^T$	$[0\ 0\ 1]^T$	rotational fixpoint
$[0\ 1\ 2]^T$	$[0\ 1\ 0]^T$	
$[0\ 2\ 2]^T$	$[0\ 2\ 2]^T$	fixpoint
$[2\ 2\ 2]^T$	$[1\ 0\ 0]^T$	
$[2\ 2\ 1]^T$	$[1\ 0\ 1]^T$	bent
$[2\ 0\ 1]^T$	$[1\ 1\ 0]^T$	
$[2\ 0\ 2]^T$	$[1\ 1\ 2]^T$	bent
$[2\ 1\ 2]^T$	$[1\ 2\ 1]^T$	rotational fixpoint
$[2\ 1\ 1]^T$	$[1\ 2\ 2]^T$	rotational fixpoint

Table 1: Value vector of ternary 1-place functions and their bent RMF spectra.

Definition 8. If \mathbf{F} is a bent function and $\mathbf{R}_f = 2\mathbf{F}$, or equivalently $\mathbf{F} = 2\mathbf{R}_f$, this constitutes a rotational fixpoint.

Lemma 5. The condition of Lemma 2, that $f(2) \neq 0$, is necessary and sufficient.

Proof: There are $3^3 = 27$ ternary 1-place functions. There are $3^2 = 9$ ternary 1-place functions such that $f(2) = 0$, which from Lemma 2 are not bent. There are 18 remaining ternary 1-place functions such that $f(2) \neq 0$, which from Lemma 2 are bent. The assertion follows. \square

Lemma 6. A necessary condition for a 2-place ternary function to have a bent Reed-Muller-Fourier spectrum is that $f(5) = f(7) = f(8) = 0$.

Proof: By inspection of a database. (See Table 3). \square

Lemma 7. There are 16 ternary bent functions on two variables that have also a bent RMF spectrum. 2 of them are fixed points. 4 of them are rotational fixed points.

Bent function	Bent RMF spectrum
$[0\ 1\ 0\ 2\ 2\ 0\ 1\ 0\ 0]^T$	$[0\ 1\ 2\ 2\ 1\ 0\ 0\ 0\ 0]^T$
$[0\ 1\ 2\ 2\ 1\ 0\ 0\ 0\ 0]^T$	$[0\ 1\ 0\ 2\ 2\ 0\ 1\ 0\ 0]^T$
$[\mathbf{0}\ \mathbf{1}\ \mathbf{1}\ \mathbf{2}\ \mathbf{0}\ \mathbf{0}\ \mathbf{2}\ \mathbf{0}\ \mathbf{0}]^T$	$[\mathbf{0}\ \mathbf{1}\ \mathbf{1}\ \mathbf{2}\ \mathbf{0}\ \mathbf{0}\ \mathbf{2}\ \mathbf{0}\ \mathbf{0}]^T$
$[2\ 0\ 1\ 2\ 1\ 0\ 0\ 0\ 0]^T$	$[1\ 1\ 0\ 0\ 2\ 0\ 2\ 0\ 0]^T$
$[2\ 2\ 0\ 0\ 1\ 0\ 1\ 0\ 0]^T$	$[1\ 0\ 2\ 1\ 2\ 0\ 0\ 0\ 0]^T$
$[2\ 2\ 1\ 0\ 2\ 0\ 2\ 0\ 0]^T$	$[1\ 0\ 1\ 1\ 1\ 0\ 2\ 0\ 0]^T$
$[\mathbf{2}\ \mathbf{1}\ \mathbf{0}\ \mathbf{1}\ \mathbf{2}\ \mathbf{0}\ \mathbf{0}\ \mathbf{0}\ \mathbf{0}]^T$	$[\mathbf{1}\ \mathbf{2}\ \mathbf{0}\ \mathbf{2}\ \mathbf{1}\ \mathbf{0}\ \mathbf{0}\ \mathbf{0}\ \mathbf{0}]^T$
$[\mathbf{2}\ \mathbf{1}\ \mathbf{1}\ \mathbf{1}\ \mathbf{0}\ \mathbf{0}\ \mathbf{1}\ \mathbf{0}\ \mathbf{0}]^T$	$[\mathbf{1}\ \mathbf{2}\ \mathbf{2}\ \mathbf{2}\ \mathbf{0}\ \mathbf{0}\ \mathbf{2}\ \mathbf{0}\ \mathbf{0}]^T$

Table 2: Value vectors of ternary 2-place bent functions with bent RMF spectra. (In bold, a fixed point; in bold-italics, rotational fixed-points).

Proof: Exhaustive search. (See Table 3). □

Table 2 shows one half of the 2-place ternary bent functions with bent RMF spectra. The other half may be obtained after scaling all entries by 2.

Definition 9. Let f and g be ternary functions in m and n variables, respectively. Then:

$$\mathbf{F} \odot \mathbf{G} := (\mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^g]^T) \oplus ([\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G}). \quad (8)$$

This operation \odot will be called *restricted tensor sum*. (Notice the relative similarity with the tensor sum (6)).

Lemma 8. Let f and g be ternary functions in m and n variables, respectively, such that \mathbf{R}_f and \mathbf{R}_g are bent. Then the RMF spectrum of $\mathbf{F} \odot \mathbf{G}$ is bent.

Proof:

$$\begin{aligned}
 \mathbf{T}(m+n) \cdot (\mathbf{F} \odot \mathbf{G}) &= \mathbf{T}(m+n) \cdot (\mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^g]^T \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G}) \\
 &= \mathbf{T}(m+n) \cdot (\mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^g]^T) \\
 &\quad \oplus \mathbf{T}(m+n) \cdot ([\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G}) \\
 &= (p-1) \langle (\mathbf{T}(m) \cdot \mathbf{F}) \otimes \mathbf{T}(n) \cdot [\mathbf{1}, \mathbf{0}^g]^T \\
 &\quad \oplus \mathbf{T}(m) \cdot [\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{T}(n) \cdot \mathbf{G} \rangle \\
 &= (p-1) \langle \mathbf{R}_f \otimes \mathbf{1}^G \oplus \mathbf{1}^F \otimes \mathbf{R}_g \rangle \\
 &= (p-1) (\mathbf{R}_f \boxplus \mathbf{R}_g).
 \end{aligned} \tag{9}$$

Since \mathbf{R}_f and \mathbf{R}_g are bent, then $\mathbf{R}_f \boxplus \mathbf{R}_g$ is also bent [8]. Furthermore, scaling by $(p-1)$ preserves the bentness. The assertion follows. \square .

Example 2. Let f and g be ternary functions of 1 and 2 variables, respectively and let their value vectors be $\mathbf{F} = [1 \ 1 \ 2]^T$ and $\mathbf{G} = [0 \ 2 \ 1 \ 0 \ 1 \ 0 \ 2 \ 0 \ 0]^T$. (Their RMF spectra are $\mathbf{R}_f = [2 \ 0 \ 2]^T$ and $\mathbf{R}_g = [0 \ 2 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0]^T$.) It will be shown that the RMF spectrum of $\mathbf{F} \odot \mathbf{G}$ is bent.

$$\begin{aligned}
 \mathbf{F} \odot \mathbf{G} &= \mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^g]^T \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G} \\
 &= [1 \ 1 \ 2]^T \otimes [\mathbf{1}, \mathbf{0}^g]^T \\
 &\quad \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes [0 \ 2 \ 1 \ 0 \ 1 \ 0 \ 2 \ 0 \ 0]^T \\
 &= [1 \ \mathbf{0}^g \ 1 \ \mathbf{0}^g \ 2 \ \mathbf{0}^g]^T \oplus [0 \ 2 \ 1 \ 0 \ 1 \ 0 \ 2 \ 0 \ 0 \ \mathbf{0}^f \mathbf{G}]^T \\
 &= [1 \ 2 \ 1 \ 0 \ 1 \ 0 \ 2 \ 0 \ 0 \ 1 \ \mathbf{0}^g \ 2 \ \mathbf{0}^g]^T.
 \end{aligned}$$

Let

$$\mathbf{H} = \begin{bmatrix} 1 & 2 & 1 & 0 & 1 & 0 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T.$$

Then $\mathbf{F} \odot \mathbf{G} = \text{vec}(\mathbf{H})$.

The RMF spectrum of $\mathbf{F} \odot \mathbf{G}$ is given by $\mathbf{T}(3) \cdot (\mathbf{F} \odot \mathbf{G}) \bmod 3$. The direct calculation gives:

$$\mathbf{R}_H = \begin{bmatrix} 2 & 1 & 2 & 2 & 0 & 0 & 0 & 0 & 2 \\ 0 & 2 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 2 & 1 & 2 & 2 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}^T.$$

It is simple to prove that vectorizing this matrix one obtains $\mathbf{R}_f \boxplus \mathbf{R}_g$, and since these spectra are bent, then, the RMF spectrum of $\mathbf{F} \odot \mathbf{G}$ is bent [8]. Furthermore, the calculation of the absolute value of the spectral coefficients of the Vilenkin-Chrestenson spectrum of \mathbf{R}_H after its complex encoding, gives $3\sqrt{3} = 3^{1.5} = 3^{n/2}$ since $n = 3$. This spectrum is flat. Hence, the RMF spectrum of $\mathbf{F} \odot \mathbf{G}$ is bent.

Lemma 9. *The operation \odot is associative.*

Proof:

From Eq. (8) it may be seen that:

$$\begin{aligned} \text{length}(\mathbf{F} \odot \mathbf{G}) &= \text{length}(\mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^g]^T) = \text{length}(\mathbf{F} \otimes \mathbf{G}) \\ &= \text{length}(\mathbf{F}) \cdot \text{length}(\mathbf{G}). \end{aligned} \quad (10)$$

Let \mathbf{V} denote the value vector of a ternary function with not necessarily the same number of variables as \mathbf{F} and \mathbf{G} . Let

$$l_1 = \text{length}(\mathbf{F} \odot \mathbf{G}) - 1 \quad \text{and} \quad l_2 = \text{length}(\mathbf{G} \odot \mathbf{V}) - 1. \quad (11)$$

Let

$$\mathbf{Q}_1 = (\mathbf{F} \odot \mathbf{G}) \odot \mathbf{V} \quad \text{and} \quad \mathbf{Q}_2 = \mathbf{F} \odot (\mathbf{G} \odot \mathbf{V}). \quad (12)$$

i)

$$\begin{aligned} \mathbf{Q}_1 &= \langle \mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^g]^T \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G} \rangle \odot \mathbf{V} \\ &= \langle \mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^g]^T \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G} \rangle \otimes [\mathbf{1}, \mathbf{0}^v]^T \oplus [\mathbf{1}, \mathbf{0}^{l_1}]^T \otimes \mathbf{V} \\ &= \mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^g]^T \otimes [\mathbf{1}, \mathbf{0}^v]^T \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G} \\ &\quad \otimes [\mathbf{1}, \mathbf{0}^v]^T \oplus [\mathbf{1}, \mathbf{0}^{l_1}]^T \otimes \mathbf{V} \\ &= \mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^v, \mathbf{0}^{gV}]^T \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G} \otimes [\mathbf{1}, \mathbf{0}^v]^T \oplus [\mathbf{1}, \mathbf{0}^{l_1}]^T \otimes \mathbf{V}. \end{aligned} \quad (13)$$

ii)

$$\begin{aligned} \mathbf{Q}_2 &= \mathbf{F} \odot \langle \mathbf{G} \otimes [\mathbf{1}, \mathbf{0}^v]^T \oplus [\mathbf{1}, \mathbf{0}^g]^T \otimes \mathbf{V} \rangle \\ &= \mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^{l_2}]^T \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes \langle \mathbf{G} \otimes [\mathbf{1}, \mathbf{0}^v]^T \oplus [\mathbf{1}, \mathbf{0}^g]^T \otimes \mathbf{V} \rangle \\ &= \mathbf{F} \otimes [\mathbf{1}, \mathbf{0}^{l_2}]^T \oplus [\mathbf{1}, \mathbf{0}^f]^T \otimes \mathbf{G} \otimes [\mathbf{1}, \mathbf{0}^v]^T \oplus [\mathbf{1}, \mathbf{0}^g, \mathbf{0}^{fG}]^T \otimes \mathbf{V}. \end{aligned} \quad (14)$$

Notice that

$$\begin{aligned} [\mathbf{1}, \mathbf{0}^f]^T \otimes [\mathbf{1}, \mathbf{0}^g]^T &= [\mathbf{1}, \mathbf{0}^g, \mathbf{0}^{fG}]^T = [\mathbf{1}, \mathbf{0}^{fG+g}]^T \\ &= [\mathbf{1}, \mathbf{0}^{fG+G-1}]^T = [\mathbf{1}, \mathbf{0}^{G(f+1)-1}]^T \\ &= [\mathbf{1}, \mathbf{0}^{GF-1}]^T = [\mathbf{1}, \mathbf{0}^{l_1}]^T. \end{aligned} \quad (15)$$

Similarly

$$[\mathbf{1}, \mathbf{0}^g]^T \otimes [\mathbf{1}, \mathbf{0}^v]^T = [\mathbf{1}, \mathbf{0}^{b_2}]^T. \quad (16)$$

Introducing (15) and (16) in (13) and (14), respectively, it becomes clear that $\mathbf{Q}_1 = \mathbf{Q}_2$, which proves that \odot is associative. \square

Remark 1. : It should be noticed that Lemma 8 (as shown in the example) opens an iterative procedure to generate functions of a high number of variables, which exhibit a bent RMF spectrum. For example let f , g , and k be ternary functions with bent RMF spectra. Let $\mathbf{J} = \mathbf{F} \odot \mathbf{G}$. From Lemma 8, \mathbf{R}_j is bent. If $\mathbf{Z} = \mathbf{F} \odot \mathbf{G} \odot \mathbf{K}$ then $\mathbf{Z} = \mathbf{J} \odot \mathbf{K}$, (since \odot is associative) and from Lemma 8, \mathbf{R}_z will be bent.

4 Classification and Characterization

The set of 486 ternary bent functions on two variables [8], which in this study are interpreted as bent RMF spectra, may be partitioned in six classes of 81 spectra each. Let the first class be called Σ_{01} comprising spectra which exhibit a $0\hat{\Delta}1$ prefix, e.g.

$$[\underline{0} \ \underline{0} \ \underline{0} \ \underline{0} \ \underline{1} \ 2 \ 0 \ 2 \ 1]^T, \text{ and } [\underline{0} \ \underline{1} \ 2 \ 2 \ 2 \ 2 \ 2 \ 1 \ 0]^T.$$

The second class, called Σ_{12} comprises all spectra obtained by adding the constant vector $\mathbf{1}^9$, (of length 9), to spectra of the class Σ_{01} , e.g. $[\underline{1} \ \underline{1} \ \underline{1} \ \underline{1} \ \underline{2} \ 0 \ 1 \ 0 \ 2]^T$, and $[\underline{1} \ \underline{2} \ 0 \ 0 \ 0 \ 0 \ 0 \ 2 \ 1]^T$, obtained from the spectra of the former class.

Adding 1 mod 3 is a spectral invariance operation [17], which preserves bentness. Notice that spectra in this class have the prefix $1\hat{\Delta}2$ as underlined in the two examples. This explains the index of Σ_{12} .

The third class is obtained by complementing all spectra of the class Σ_{12} . The ternary complement preserves the value 1 and exchanges the values 0 and 2. Therefore, this class will be called Σ_{10} . If \mathbf{F} is the value vector of a ternary function, the value vector of the complement of \mathbf{F} is given by $2\mathbf{F} \oplus \mathbf{2}^9$. Spectra in this class, corresponding to the previous examples are e.g. $[\underline{1} \ \underline{1} \ \underline{1} \ \underline{1} \ \underline{0} \ 2 \ 1 \ 2 \ 0]^T$ and $[\underline{1} \ \underline{0} \ 2 \ 2 \ 2 \ 2 \ 2 \ 0 \ 1]^T$.

There are no repetitions of spectra of the class Σ_{12} . Let \mathbf{R} be a bent spectrum in the class Σ_{01} . Then $\mathbf{R} \oplus \mathbf{1}^9$ will belong to Σ_{12} and its complement may formally be expressed as $2(\mathbf{R} \oplus \mathbf{1}^9) \oplus \mathbf{2}^9 = 2\mathbf{R} \oplus \mathbf{2}^9 \oplus \mathbf{2}^9 = 2\mathbf{R} \oplus \mathbf{1}^9$. This is not the structure of the spectra in Σ_{12} . Spectra in this class have the prefix $1\hat{\Delta}0$, as may be seen in the examples. Therefore, complementing the spectra in Σ_{12} generates a disjoint class. Moreover, $2\mathbf{R} \oplus \mathbf{1}^9$ is bent since both scaling by 2 and adding 1 in mod 3 are spectral invariance operations that preserve bentness. The remaining three classes

are obtained scaling by 2 all entries of the previous three classes. Recall that scaling by 2 is a spectral invariance operation and the RMF transform preserves scaling (Lemma 2, ii). Table 3 shows the ternary functions and the corresponding RMF bent spectra associated to the class Σ_{01} . The following question will be studied below. Which is the relationship among the ternary functions whose bent RMF spectra belong to the classes described above?

Definition 10. *The sum mod 3 of all elements of the value vector of a ternary function is called its modular weight possibly in analogy to the Hamming weight of Boolean functions.*

Lemma 10. *Let \mathbf{F}_0 be the value vector of a 2-place ternary function which has a bent RMF spectrum \mathbf{R}_{f_0} in the class Σ_{01} . If the bent spectrum $\mathbf{R}_{f_0} \oplus \mathbf{1}^9$ is constructed and its generating function is called \mathbf{F}_1 , this function will belong to the Σ_{12} class. The following holds:*

$$\mathbf{F}_1 = \mathbf{F}_0 \oplus 2 \cdot [1, \mathbf{0}^8]^T. \quad (17)$$

Proof:

$$\begin{aligned} \mathbf{F}_1 &= \mathbf{T}(2)(\mathbf{R}_{f_0} \oplus \mathbf{1}^9) = \mathbf{T}(2)\mathbf{R}_{f_0} \oplus \mathbf{T}(2)\mathbf{1}^9 \\ &= \mathbf{F}_0 \oplus \mathbf{T}(2) \cdot \mathbf{1}^9 = \mathbf{F}_0 \oplus \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 \\ 2 & 1 & 0 & 2 & 1 & 0 & 2 & 1 & 0 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\ &= \mathbf{F}_0 \oplus 2 \cdot [1, \mathbf{0}^8]^T \pmod{3}. \end{aligned}$$

Corollary 1. *If \mathbf{F}_0 is a bent function, then \mathbf{F}_1 is not bent, since a necessary condition for a 2-place ternary function to be bent is that its modular weight is congruent with 0 in mod 3 [8]. This condition is obviously not satisfied by \mathbf{F}_1 .*

Lemma 11. *Let \mathbf{F}_0 be the value vector of a 2-place ternary function which has a bent RMF spectrum \mathbf{R}_{f_0} in the class Σ_{01} . If the complement of the bent spectrum $\mathbf{R}_{f_0} \oplus \mathbf{1}^9$ is constructed and its generating function is called \mathbf{F}_2 , this function will belong to the class Σ_{10} . The following holds:*

$$\mathbf{F}_2 = 2 \cdot \mathbf{F}_0 \oplus 2 \cdot [1, \mathbf{0}^8]^T. \quad (18)$$

Proof:

$$\begin{aligned}\mathbf{F}_2 &= \mathbf{T}(2)\langle 2(\mathbf{R}_{f_0} \oplus \mathbf{1}^9) \oplus \mathbf{2}^9 \rangle = \mathbf{T}(2)\langle 2\mathbf{R}_{f_0} \oplus \mathbf{2}^9 \oplus \mathbf{2}^9 \rangle = \mathbf{T}(2)\langle 2\mathbf{R}_{f_0} \oplus \mathbf{1}^9 \rangle \\ &= 2\mathbf{T}(2)\mathbf{R}_{f_0} \oplus \mathbf{T}(2)\mathbf{1}^9 = 2\mathbf{F}_0 \oplus 2 \cdot [1, \mathbf{0}^8]^T.\end{aligned}$$

Corollary 2. *If \mathbf{F}_0 is a bent function, then \mathbf{F}_2 is not bent, for the same reasons as in Corollary 1.*

Bentness of functions in the first three classes is preserved in the last three classes, since these classes are obtained scaling by 2 all entries of the former ones and this scaling represents a spectral invariance operation that preserves bentness. In this case, the bentness of a function and the bentness of its RMF spectrum. Moreover, the RMF spectrum preserves scaling by 2. There is no direct relationship among bent functions belonging to the first three classes or to the last three classes. A weak relationship is the following: let f_0 belong to Σ_{01} and let its modular weight be 2. If f_0 is used to generate a function f_1 in Σ_{12} , then, with Eq. (19), the modular weight of f_1 will be 0. Since this is a necessary condition for bentness, then f_1 is only a candidate to be bent. Similarly, if f_0 has a modular weight 1 and is used to generate a function f_2 in the class Σ_{10} , then, with Eq. (20) the modular weight of f_2 will be 0, from where f_2 becomes only a candidate to be bent.

Lemma 12. *A necessary condition for a 2-place ternary function f to be both bent and have a bent RMF spectrum is the following:*

$$f(6) \oplus \bigoplus_{i=0}^4 f(i) \equiv 0 \pmod{3}. \quad (19)$$

Proof: Follows from the above Lemma 6 and the modular weight condition mentioned in Corollary 1. \square

Lemma 13. *If in Table 3 two ternary functions differ only in $f(6)$ and the difference is $k \in \{1, 2\}$, then their RMF bent spectra only differ by $2k$ in $R(6)$, $R(7)$, and $R(8)$.*

Proof:

Let \mathbf{F}_1 and \mathbf{F}_2 be ternary functions in Table 3, and let

$$\mathbf{K} = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ k \ 0 \ 0]^T,$$

where $k \in \{1, 2\}$. If $\mathbf{F}_2 = \mathbf{F}_1 \oplus \mathbf{K}$, then (with Lemma 3), $\mathbf{R}_{f_2} = \mathbf{R}_{f_1} \oplus \mathbf{T}(2) \cdot \mathbf{K}$.

A direct calculation shows that:

$$\begin{aligned}
 \mathbf{T}(2) \cdot \mathbf{K} &= (\mathbf{T}(2) \cdot [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ k \ 0 \ 0]^T) \\
 &= \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 \\ 2 & 1 & 0 & 2 & 1 & 0 & 2 & 1 & 0 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ k \\ 0 \\ 0 \end{bmatrix} \\
 &= [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 2k \ 2k \ 2k]^T.
 \end{aligned}$$

Calculations in mod 3. The assertion follows. \square

It may be noticed that the situation considered in Lemma 12 occurs quite frequently between neighbor functions in the Σ_{01} class, as may be seen in Table 3. In this table, shown in **bold**, are elements of the value vector of bent functions. In *italics* and **bold**, a fixpoint.

5 Conclusions

Ternary functions which have a bent RMF spectrum have been studied. Necessary conditions that ternary functions of 1 and 2 variables must satisfy to have bent RMF spectra are given. If $n = 2$, there are 486 RMF bent spectra, but only 16 ternary bent functions satisfy the conditions to have a bent RMF spectrum. 2 of them are fixed points and 4 are rotational fixed points. Recall that in the general case (i.e. not necessarily bent ternary RMF spectra) if $n = 2$ there are 243 fixed points and if $n = 3$ there are 313 fixed points [11]. The study of the ternary functions having a bent RMF spectrum gave origin to the concept of rotational fixed points. Two rotational fixed points were found in the class Σ_{12} . This will again appear when this class is scaled by 2. These fixed points will not appear again in the second and third class since the spectra will be shifted by 1 and 2, but according to Lemmas 10 and 11, the functions have a different transformation. Extending the present study to the case of 3-place ternary functions can hardly be based on exhaustive search, as done for $n = 1$ and $n = 2$, since when $n = 3$, there are e.g. 303,264 ternary bent functions (of degree 3) [17]. However, Lemma 8 provides an iterative construction method to obtain ternary functions on a large number of variables, such that their respective RMF spectra are bent.

ternary function	bent RMF spectrum	ternary function	bent RMF spectrum
0 0 0 0 2 0 0 0 0	0 0 0 0 1 2 0 2 1	0 1 2 2 0 0 2 0 0	0 1 0 2 0 2 2 0 2
0 0 0 0 2 0 2 0 0	0 0 0 0 1 2 1 0 2	0 1 2 2 2 0 1 0 0	0 1 0 2 1 1 0 0 1
0 0 0 0 2 0 1 0 0	0 0 0 0 1 2 2 1 0	0 1 2 2 2 0 0 0 0	0 1 0 2 1 1 1 1 2
0 0 0 1 1 0 2 0 0	0 0 0 1 0 2 0 1 2	0 1 2 2 1 0 1 0 0	0 1 0 2 2 0 0 2 2
0 0 0 1 1 0 1 0 0	0 0 0 1 0 2 1 2 0	0 1 2 2 1 0 0 0 0	0 1 0 2 2 0 1 0 0
0 0 0 1 1 0 0 0 0	0 0 0 1 0 2 2 0 1	0 1 1 0 1 0 0 0 0	0 1 1 0 0 2 0 2 0
0 0 0 1 2 0 2 0 0	0 0 0 1 2 0 0 2 1	0 1 1 0 1 0 2 0 0	0 1 1 0 0 2 1 0 1
0 0 0 1 2 0 1 0 0	0 0 0 1 2 0 1 0 2	0 1 1 0 0 0 2 0 0	0 1 1 0 1 1 1 2 2
0 0 0 1 2 0 0 0 0	0 0 0 1 2 0 2 1 0	0 1 1 0 0 0 1 0 0	0 1 1 0 1 1 2 0 0
0 0 2 0 0 0 2 0 0	0 0 1 0 0 1 1 1 2	0 1 1 0 2 0 0 0 0	0 1 1 0 2 0 0 0 2
0 0 2 0 0 0 1 0 0	0 0 1 0 0 1 2 2 0	0 1 1 0 2 0 2 0 0	0 1 1 0 2 0 1 1 0
0 0 2 0 2 0 0 0 0	0 0 1 0 1 0 0 2 2	0 1 1 1 2 0 2 0 0	0 1 1 1 0 1 0 0 2
0 0 2 0 2 0 1 0 0	0 0 1 0 1 0 2 1 1	0 1 1 1 2 0 0 0 0	0 1 1 1 0 1 2 2 1
0 0 2 0 1 0 0 0 0	0 0 1 0 2 2 0 1 0	0 1 1 1 1 0 2 0 0	0 1 1 1 1 0 0 2 0
0 0 2 0 1 0 1 0 0	0 0 1 0 2 2 2 0 2	0 1 1 1 1 0 0 0 0	0 1 1 1 1 0 2 1 2
0 0 2 1 1 0 1 0 0	0 0 1 1 0 0 1 2 1	0 1 1 1 0 0 2 0 0	0 1 1 1 2 2 0 1 1
0 0 2 1 1 0 0 0 0	0 0 1 1 0 0 2 0 2	0 1 1 1 0 0 1 0 0	0 1 1 1 2 2 1 2 2
0 0 2 1 0 0 2 0 0	0 0 1 1 1 2 0 0 1	0 1 1 2 0 0 1 0 0	0 1 1 2 0 0 0 1 1
0 0 2 1 0 0 1 0 0	0 0 1 1 1 2 1 1 2	0 1 1 2 0 0 2 0 0	0 1 1 2 0 0 2 0 0
0 0 2 1 2 0 1 0 0	0 0 1 1 2 1 1 0 0	0 1 1 2 2 0 0 0 0	0 1 1 2 1 2 1 1 0
0 0 2 1 2 0 0 0 0	0 0 1 1 2 1 2 1 1	0 1 1 2 2 0 2 0 0	0 1 1 2 1 2 2 2 1
0 0 2 2 2 0 1 0 0	0 0 1 2 0 2 0 2 2	0 1 1 2 1 0 0 0 0	0 1 1 2 2 1 1 0 1
0 0 2 2 2 0 0 0 0	0 0 1 2 0 2 1 0 0	0 1 1 2 1 0 2 0 0	0 1 1 2 2 1 2 1 2
0 0 2 2 1 0 1 0 0	0 0 1 2 1 1 0 1 0	0 1 0 0 1 0 0 0 0	0 1 2 0 0 0 0 2 1
0 0 2 2 1 0 0 0 0	0 0 1 2 1 1 1 2 1	0 1 0 0 1 0 2 0 0	0 1 2 0 0 0 1 0 2
0 0 2 2 0 0 1 0 0	0 0 1 2 2 0 0 0 1	0 1 0 0 1 0 1 0 0	0 1 2 0 0 0 2 1 0
0 0 2 2 0 0 2 0 0	0 0 1 2 2 0 2 2 0	0 1 0 0 2 0 0 0 0	0 1 2 0 2 1 0 0 0
0 1 2 0 1 0 0 0 0	0 1 0 0 0 1 0 2 2	0 1 0 0 2 0 2 0 0	0 1 2 0 2 1 1 1 1
0 1 2 0 1 0 1 0 0	0 1 0 0 0 1 2 1 1	0 1 0 0 2 0 1 0 0	0 1 2 0 2 1 2 2 2
0 1 2 0 0 0 2 0 0	0 1 0 0 1 0 1 2 1	0 1 0 1 2 0 2 0 0	0 1 2 1 0 2 0 0 0
0 1 2 0 0 0 1 0 0	0 1 0 0 1 0 1 2 1	0 1 0 1 2 0 1 0 0	0 1 2 1 0 2 0 0 0
0 1 2 0 2 0 0 0 0	0 1 0 0 2 2 0 0 1	0 1 0 1 2 0 2 0 0	0 1 2 1 0 2 2 2 2
0 1 2 0 2 0 1 0 0	0 1 0 0 2 2 0 0 1	0 1 0 1 1 0 2 0 0	0 1 2 1 0 2 2 2 2
0 1 2 1 2 0 1 0 0	0 1 0 1 0 0 1 1 2	0 1 0 1 1 0 1 0 0	0 1 2 1 1 1 1 0 2
0 1 2 1 2 0 0 0 0	0 1 0 1 0 0 2 2 0	0 1 0 1 1 0 0 0 0	0 1 2 1 1 1 2 1 0
0 1 2 1 1 0 1 0 0	0 1 0 1 1 2 1 0 0	0 1 0 2 2 0 1 0 0	0 1 2 2 1 0 0 0 0
0 1 2 1 1 0 0 0 0	0 1 0 1 1 2 2 1 1	0 1 0 2 2 0 0 0 0	0 1 2 2 1 0 1 1 1
0 1 2 1 0 0 2 0 0	0 1 0 1 2 1 0 1 0	0 1 0 2 2 0 2 0 0	0 1 2 2 1 0 2 2 2
0 1 2 1 0 0 1 0 0	0 1 0 1 2 1 1 2 1	0 1 0 2 1 0 1 0 0	0 1 2 2 2 2 0 2 1
0 1 2 2 0 0 1 0 0	0 1 0 2 0 2 0 1 0	0 1 0 2 1 0 0 0 0	0 1 2 2 2 2 1 0 2
0 1 2 2 0 0 2 0 0	0 1 0 2 0 2 2 0 2		

Table 3: Value vectors of 2-place ternary functions and their bent RMF spectra in the class Σ_{01} .

Finally, it should be recalled that the RMF transform for the binary case was introduced by J.E. Gibbs in 1977 [2] under the name Instant Fourier transform. The term Fourier was justified by the properties of the transform corresponding to the properties of the Fourier representations in classical mathematical analysis. The term Instant came from the simplicity of computing this transform by a network consisting of AND and EXOR gates as it was illustrated by an example in [2]. The RMF and the RM transform in the binary case coincide [4], although this was not observed in [2], since the research leading to these concepts came from different areas and was driven by fairly separate motivations.

References

- [1] Carlet, C., Mesnager, S., Four decades of research on bent functions, *Des. Codes Cryptogr.*, Vol. 78, 5-50, 2016.
- [2] Gibbs, J.E.: Instant Fourier transform, *Electronics Letters*, 13, (5), 122-123, 1977.
- [3] Horn R.A., Johnson Ch.R.: *Topics in Matrix Analysis*. Cambridge University Press, 1991.
- [4] Karpovsky, M.G., StankoviĀĀ, R.S., Astola, J.T.: *Spectral Logic and its Applications for the Design of Digital Devices*. John Wiley and Sons, 2008.
- [5] Kumar P.V., Scholz R.A., Welch L.R.: Generalized bent functions and their properties. *Jr. Combinatorial Theory Series A*, 40, (1): 90-107, 1985.
- [6] Mesnager S.: *Bent Functions. Fundamentals and Results*, Springer, New York, 2016.
- [7] Moraga C.: On a property of the Reed-Muller-Fourier transform, *Facta Universitatis, Series Electronic and Energetics*, 31, (2), 303-311, 2018.
- [8] Moraga C., StankoviĀĀ M.M., StankoviĀĀ R.S., StojkoviĀĀ S.: A contribution to the Study of Multiple-Valued Bent functions. *Proc. 43rd Int. Symposium on Multiple-Valued Logic*, 340-345. IEEE Press, 2013.
- [9] Moraga C., StankoviĀĀ R.S., StankoviĀĀ M.M.: The Pascal triangle (1654), the Reed-Muller-Fourier Transform (1992), and the Discrete Pascal Transform (2005). In *Proc. 46th Int. Symposium on Multiple-valued Logic*. 229-234. Sapporo, Japan. IEEE Press, 2016.
- [10] Moraga C., StankoviĀĀ M.M., StankoviĀĀ R.S.: Contribution to the study of ternary functions with a bent Reed-Muller spectrum. In *Proc. 45th Int. Symposium on Multiple-valued Logic*, 133-138, Waterloo, Canada, IEEE Press, 2015.
- [11] Moraga C., StankoviĀĀ M.M., StankoviĀĀ R.S., StojkoviĀĀ S.: On Fixed Points of the Reed-Muller-Fourier Transform. In *Proc. 47th Int. Symposium on Multiple-valued Logic*. 55-60. Novi Sad, Serbia, IEEE Press, 2017.
- [12] Moraga C., StankoviĀĀ R.S.: The Reed-Muller-Fourier transform applied to pattern analysis. *LNCS 10672*, 254-261, Springer, ISBN 978-3-319-74726-2, 2018.

- [13] Moraga C., StankoviĀ R.S., Astola J.T.: On the Reed-Muller-Fourier spectrum of multiple-valued rotation symmetric functions. In *Proc. 48th Int. Symposium Multiple-valued Logic*, Linz, Austria, 241-246, IEEE-Press, 2018.
- [14] Moraga C., StankoviĀ R.S., StankoviĀ M.M.: Generalized Permutations and Ternary Bent Functions. *ArXiv* abs. 1912.08615, 2019.
- [15] Moraga C., StankoviĀ M.M., StankoviĀ R.S., StojkoviĀ S.: Methods to generate Multiple-valued Bent Functions of an odd number of Variables. In *Proc. 50th Int. Symposium Multiple-valued Logic*, 70-75, Miyazaki, Japan. IEEE Press, 2020.
- [16] Rothaus O.S. On \mathbb{F}_3 -Bent Functions. *Jr. of Combinatorial Theory (A)*, 20, 300-305, 1976.
- [17] StankoviĀ, M.M., Moraga C., StankoviĀ, R.S.: Spectral invariance operations for the construction of ternary bent functions. *Journal of Applied Logics*. To be published, 2022.
- [18] StankoviĀ, R.S.: Some remarks on Fourier transforms and differential operators for digital functions, *Proc. 22nd Int. Symp. on Multiple-valued Logic*, Sendai, Japan, 365-370. DOI:10.1109/ ISMVL.1992.186818, 1992.
- [19] StankoviĀ R.S., Moraga C.: Reed-Muller-Fourier representations of multiple-valued functions over Galois fields of prime cardinality. In Kebschull, U., Schubert, E., Rosentiel, W., (Eds.), *Proc. IFIP WG10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, 115-124, 1993.
- [20] StankoviĀ, R.S., StankoviĀ, M., Moraga, C., Sasao, T.: Calculation of Reed-Muller-Fourier coefficients of multiple-valued functions through multiple place decision diagrams. *Proc. 24th Int. Symp. on Multiple-valued Logic*, 82-88. IEEE Press, 1994.
- [21] StankoviĀ, R.S., Moraga, C., Reed-Muller-Fourier representations of multiple-valued functions, in StankoviĀ, R.S., StojkoviĀ, M.R., StankoviĀ, M.S., (Eds.) *Recent Developments in Abstract Harmonic Analysis with Applications in Signal Processing*, Nauka, Belgrade, Elektronski Fakultet, NiĀ, 205-216, 1996.
- [22] StankoviĀ, R.S., Astola J.T., Moraga, C.: *Representation of Multiple-Valued Logic Functions*, Morgan & Claypool, San Rafael, USA, 2012.
- [23] StankoviĀ, R.S.: The Reed-Muller-Fourier Transform – Computing methods and factorizations. In R. Seising, H. Allende-Cid, (Eds.), *Claudio Moraga: A Passion for Multi-Valued Logic and Soft Computing*, 121-151, Springer, 2017.
- [24] StankoviĀ, R.S., StankoviĀ M.M., Astola J.T., Moraga C.: Gibbs characterization of binary and ternary bent functions. In *Proc. 46th Int. Symposium Multiple-valued Logic*. 205-210. Sapporo, Japan. IEEE Press, 2016.
- [25] Tokareva, N.: *Bent Functions. Results and Applications to Cryptography*. Elsevier – Academic Press, Amsterdam, 2015.
- [26] Vilenkin N. Ya. Agaev G. N., Dzafarli G. M.: Towards a theory of multiplicative orthogonal systems of functions, *DAN Azerb. SSR*, 18, No. 9, 3-7, 1962.
- [27] Waldhauser, T.: On the number of fixed points of the Reed-Muller-Fourier transform, *Proc. 48th Int. Symposium Multiple-Valued Logic*, 229-234, IEEE Press, 2018.

- [28] Waldhauser, T.: On eigenvectors of the Pascal and Reed-Muller-Fourier transforms, *Acta Cybern.*, Vol. 23, No. 3, 959-979, 2018.
- [29] Waldhauser, T., Bases for the space of fixed points of the Reed-Muller-Fourier Transform, *Journal of Multiple Valued Logic and Soft Computing*, Vol. 34, No. 3-4, 239-259, 2020.