School of Electrical, Electronic & Computer Engineering

Mixed Radix Design Flow for Security Applications

A.Rafiev

Technical Report Series

NCL-EECE-MSD-TR-2011-175

September 2011

Contact:

ashur.rafiev@ncl.ac.uk

Supported by EPSRC grant GR/F016786/1

NCL-EECE-MSD-TR-2011-175

School of Electrical, Electronic & Computer Engineering,

Merz Court,

Newcastle University,

Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Abstract

The purpose of secure devices, such as smartcards, is to protect sensitive information against software and hardware attacks. Implementation of the appropriate protection techniques often implies non-standard methods that are not supported by the conventional design tools. In the recent decade the designers of secure devices have been working hard on customising the workflow. The presented research aims at collecting the up-to-date experiences in this area and create a generic approach to the secure design flow that can be used as guidance by engineers.

Well-known countermeasures to hardware attacks imply the use of specific signal encodings. Therefore, multi-valued logic has been considered as a primary aspect of the secure design. The choice of radix is crucial for multi-valued logic synthesis. Practical examples reveal that it is not always possible to find the optimal radix when taking into account actual physical parameters of multi-valued operations. In other words, each radix has its advantages and disadvantages. Our proposal is to synthesise logic in different radices, so it could benefit from their combination.

With respect to the design opportunities of the existing tools and the possibilities of developing new tools that would fill the gaps in the flow, two distinct design approaches have been formed: conversion driven design and pre-synthesis.

The conversion driven design approach takes the outputs of mature and time-proven electronic design automation (EDA) synthesis tools to generate mixed radix datapath circuits in an endeavour to investigate the added relative advantages or disadvantages. An algorithm underpinning the approach is presented and formally described together with secure gate-level implementations. The obtained results are reported showing an increase in power consumption, thus giving further motivation for the second approach.

The pre-synthesis approach is aimed at improving the efficiency by using multi-valued logic synthesis techniques to produce an abstract component-level circuit before mapping it into technology libary. Reed-Muller expansions over Galois field arithmetic have been chosen as a theoretical foundation for this approach. In order to enable the combination of radices at the mathematical level, the multi-valued Reed-Muller expansions have been developed into mixed radix Reed-Muller expansions. The goals of the work is to estimate the potential of the new approach and to analyse its impact on circuit parameters down to the level of physical gates. The benchmark results show the approach extends the search space for optimisation and provides information on how the implemented functions are related to different radices.

The theory of two-level radix models and corresponding computation methods are the primary theoretical contribution. It has been implemented in RMMixed tool and interfaced to the standard EDA tools to form a complete security-aware design flow.

Acknowledgements

First of all, I would like to thank my mother, Nina, for her love and patience.
Even from far away she has been supporting me every day.

I am very grateful to my supervisors, Alex Yakovlev and Albert Koelmans. I would have never been able to succeed in my research without their help and guidance.

Also many thanks to my friends and colleagues – Andrey Mokhov, Arseniy Alekseyev, Julian Murphy, Frank Burns, Danil Sokolov, Stanislavs Golubcovs, and Ivan Poliakov – for many useful and inspiring ideas, criticism, support, and for just being very nice.

I kindly thank Michael Keller and Anne Harrow for the spiritual support and encouragement they have been giving me throughout these years.

This research was supported by the EPSRC grant GR/F016786/1 (SURE) and the ORS Awards Scheme.

Contents

List of Figures

List of Tables

List of Publications

Journal Papers

· A. Rafiev, A. Mokhov, F. P. Burns, J. P. Murphy, A. Koelmans, and A. Yakovlev, "Mixed radix Reed-Muller expansions," *IEEE Transactions on Computers*, 2011. Accepted for publication.

Refereed conference papers

· A. Rafiev, J. Murphy, D. Sokolov, and A. Yakovlev, "Conversion driven design of binary to mixed radix circuits," in *Proc. to ICCD*, 2008.

· A. Rafiev, J. P. Murphy, and A. Yakovlev, "Quaternary Reed-Muller expansions of mixed radix arguments in cryptographic circuits," in *Proc. to 39th International Symposium on Multi-Valued Logic, ISMVL 2009*, 2009.

· A. Rafiev, J. Murphy, and A. Yakovlev, "Secure design flow for asynchronous multivalued logic circuits," in *Proc. to 40th International Symposium on Multi-Valued Logic, ISMVL 2010*, May 2010.

Other conferences

· A. Rafiev, J. Murphy, D. Sokolov, and A. Yakovlev, "Bitwise gate grouping algorithm for mixed radix conversion," in *20th UK Asynchronous Forum*, 2008.

· A. Rafiev, J. P. Murphy, A. Yakovlev. "Higher Radix and Mixed Radix Logic in Secure Devices From the Design Automation Perspective". in *Proc. to e-Smart*

Conference, SmartEvent '09, September 2009

Technical reports and tools

· A. Rafiev, J. Murphy, D. Sokolov, and A. Yakovlev, "Investigating gate grouping algorithms for mixed radix conversion," tech. rep., Newcastle University, May 2008.

· A. Rafiev, J. Murphy, and A. Yakovlev, "RTL implementations of GF(2) and GF(4) arithmetic components," tech. rep., Newcastle University, 2008.

· RMMixed MVL synthesis tool. http://async.org.uk/sure/rmmixed.

Chapter 1

Introduction

In the modern world, digital devices take an important part in every aspect of our every day lives, including such sensitive areas as healthcare and finance. Large amount of personal and confidential information is stored online, and must be securely transferred between the client and the server. Financial operations are done "by wire", and every bank transaction is performed by some kind of electronic equipment. No doubt, all these aspects imply a great deal of trust in the devices involved. Ideally, we desire a system where the confidential information cannot be retrieved by an unauthorised person. In reality, this requires a huge effort to maintain the acceptable level of protection. Secure technologies have to continuously move forward chased by the constantly evolving techniques of the hackers.

Security solutions in the past were predominantly at the software level. The classic approach to cryptanalysis requires knowledge of ciphertext and, if available, plaintext. Cryptographic algorithms developed accordingly, minimising the possibility of getting the secret key by comparing the ciphertext to the plaintext (differential cryptanalysis [32]). With increasing complexity of the new encryption algorithms, more interest has been shown towards implementing cryptographic functions in hardware [60]. Circuit-level security, appraised for higher performance, has been also considered to be more secure. However, this has raised new problems as well.

Working entirely at the mathematical level, the traditional cryptography does not

take into account possible flaws of the actual implementations. Running on the real hardware, a cryptosystem dissipates different kind of information that may indirectly disclose the data being processed. For example, certain bits of a message may take a slightly longer time to compute or cause variations in the device's power consumption. The type of attacks that rely on this data is called side-channel attacks.

The major concern about side-channel attacks starts in 1998 with the discovery of the power analysis [39]. The idea of simple power analysis (SPA) is to examine device's power curve during the normal operation w.r.t. the cryptographic algorithm and the data being processed. For example, the operation of multiplication normally consumes more power than addition, hence the RSA algorithm [61, 63] can expose the private key while branching between these operations. The differential power analysis (DPA) uses power samples from multiple runs and performs statistical analysis in order to find the correlation and reveal the key. DPA allows breaking the commonly used algorithms in relatively short time. For instance, at the mathematical level, DES is perfectly resistant to differential cryptanalysis and takes exponential time to crack, while DPA requires only 1000 power samples [40]. The alarming efficiency of this attack drew the attention of both smartcard vendors and the cryptographic community, and even featured in an article in the New York Times [74]. The new standard on secure devices has been altered accordingly [5].

More details on DPA and countermeasures are given in Chapter 2. Among the number of described countermeasures, the presented work has been focused on power balancing and asynchronous system design only. The idea of power balancing is to encode and process the data signal in such a way that the operation over it will produce data independent power consumption due to the uniform switching of gates. Balanced encodings also imply more than 2 states of the signal, which leads to the multi-valued logic synthesis approach. In asynchronous designs the clock is replaced with handshake signals. Fuzzy circuit timing characteristics makes it difficult to sample power curves required for DPA.

The conventional industrial electronic design automation (EDA) tools, such as

Synopsys [7], Cadence [4], Magma [6] are very powerful, but they are targetted towards the general purpose devices and commonly used technologies. These tools are focusing on the primary issues of the market in order to deliver high quality solutions to the most of their customers. As a result, this approach often excludes many other aspects, such as multi-valued logic synthesis and asynchronous design. Industrial tools can be compared to an assembly line that can produce thousands of typical items with a push of a button. However, it's probably not the best choice if one needs something *special*.

Consequently, the development of secure devices requires much more design effort and should employ the methodologies beyond the "casual" electronic design. Some research has been made in an attempt to adjust the design flow [70, 11], but simple 'tweaks' do not give the best efficiency. In order to be ahead of the hackers in their technological pursuit, the designers need a very good tool support. The research presented in this thesis aims at formulating the unified design flow, filling in the gaps in the conventional design automation with the aid of the existing tools and development of the new ones.

1.1 Motivation

Design flows used in industry are generally based on binary logic with a single bit of data represented as a single wire. This representation of data is called "single-rail" encoding. Multi-valued logic (MVL) synthesis tools also exist, e.g. MVSIS [27], and they are typically used to efficiently produce logic from higher-radix specifications, while the technology is still single-rail and processed using the standard flow and tools. The work described in the thesis, however, deals with the application areas where a simple binary encoding of data is either not applicable or gives poor results in terms of the delivered properties of the implementation.

Important features of certain encodings are exposed when used in connection with appropriate protocols. For example, single-rail is used with bundled data protocol [68] for asynchronous system design. From the application perspective, it is convenient

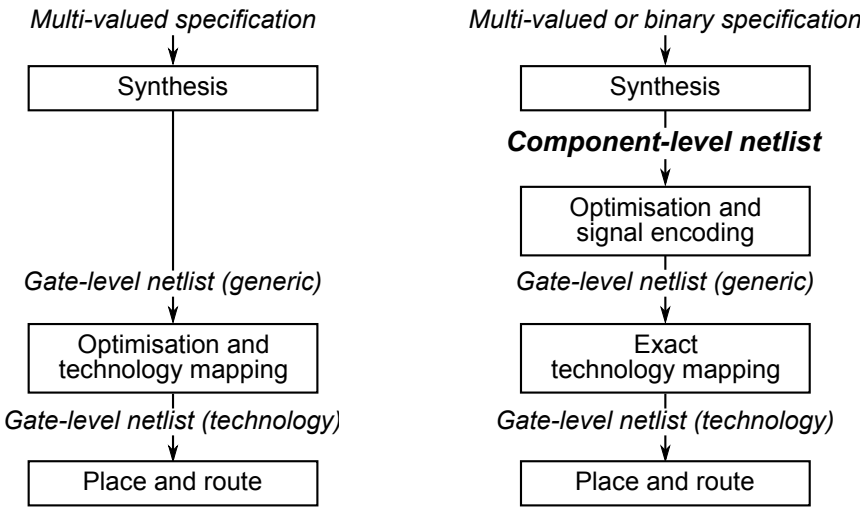
to consideer an encoding-protocol combination as a whole in order to explore their properties and possibilities.

Thus, m-of-n encoding combined with a spacer protocol is beneficial for security [17, 76], but for the price of increased area and power consumption. Chapter 2 outlines the problem; however, the discussion on the efficiency of certain countermeasures against the side-channel attacks is out of scope of this thesis. We are mostly interested in the fact that m-of-n codes imply MVL, which is problematic to design using the standard techniques.

It is also important that this particular type of encoding is not the only possible application area of MVL, and therefore the presented research. Possible use cases may include current-mode [13] or phase encoding [20], or any other protocol displaying the properties that single-rail does not have. On-chip interconnects [14] and low-power [33] devices are well-established applications for MVL that use special data representation at the physical level. The application of MVL may even go beyond microelectronics and, possibly, enter such areas as biocomputing [10].

When the problem of using an encoding arises, the designers struggle to use the existing methodologies, while in fact a brand new design flow is warranted. This implies the need for an "application-driven" design flow. Figure 1.1 illustrates the difference between the approaches, crucial for understanding the motivation behind our research.

Typically, in a general purpose flow, the intermediate structure of MVL does not matter, because it produces the gate-level (binary) netlist that can be henceforth fed to a standard design compiler, where it is compacted and mapped into the certain technology. In contrast, an application-driven design flow explicitly defines an intermediate synthesis state known as the component-level netlist – a netlist of abstract arithmetic operations. The component-level netlist is an important link that carries along the radix information, which is essential for correct optimisation and data encoding. The consequent technology mapping does not alter the structure of the circuit; therefore, even if the final circuit is mapped into binary logic gates, compliance with the protocols



(a) typical general purpose flow (b) application-driven flow

Figure 1.1: Understanding the difference between general purpose MVL and application-driven MVL synthesis flows

is preserved.

The physical implementation of arithmetic components directly depends on the encoding, thus the radix of signals, and has a strong impact on the characteristics of the circuit. Therefore the right choice of the radix is highly important; however, it is not always possible to find the globally optimal solution. Lower order radices require less logic to compute, and higher radices reduce the number of interconnects [45]. Our examples also show that this logic/interconnect trade-off is not evenly distributed among the arithmetic operations. Certain operations, if implemented with respect to the application requirements, have better physical characteristics in the lower radix logic, while other operations are more efficient in the higher radices. Consequently, when all the operations are combined, a single optimal radix cannot be worked out.

Hence, a methodology to partition the synthesised circuit into areas of different radices, whilst preserving the integrity and the properties of the original function, would be of great advantage. The ability to accommodate arithmetic components of different radices within a single circuit may provide the opportunity to incorporate the

benefit from each, and also provide another dimension in the logic/interconnect trade-off. This describes a design philosophy where the radix is not a single fundamental idea but just a "tool" for computation, so the designer can use different "tools" together in order to achieve the best results.

A straightforward solution to mix radices within a single circuit is to use signal conversion, as described in Chapter 3. For example, signal conversion logic can split quaternary signals into pairs of binary signals where appropriate, and vice versa. Conversion can be performed in a rather sophisticated way [71]. The main interest in the conversion approach is the possibility to apply it on top of the existing designs with a minimum computational cost. A major drawback, however, is that the structure of the circuit previously optimised for a certain radix becomes less efficient when converted to a different radix.

The problem can be avoided if mixed radices are applied directly at the logic synthesis stage. The proposal is to develop a mathematical theory for mixing radices in such a way that the radix conversion is done "by construction". Reed-Muller expansions over Galois fields can be chosen as a foundation for the approach. Since most cryptographic algorithms are based on Galois field arithmetic, we expect these functions to be efficiently mapped into Reed-Muller expansions.

Among a number of MVL synthesis techniques, Reed-Muller expansions over the Galois field of radix 4 are of great interest and have been developed for a number of years. The history of multi-valued Reed-Muller expansions started with evolving functional binary decision diagrams into the theory of Galois switching functions by applying finite field algebra [29, 52]. Later on, this approach has been developed into fixed and mixed polarity quaternary Reed-Muller expansions [28] – acknowledged by MVL research community due to their efficiency and testability. However, the high computational cost has prevented this approach from wide-spread and practical use. Thus, the later research has been focused mostly on optimising the computational algorithms [24, 59, 36, 25] and logic minimisation [34]. An attempt to implement multi-valued Reed-Muller expansions in CMOS has been made as well [81].

Conveniently for the application-driven flow, a Reed-Muller expansion is produced in the form of an arithmetic expression that can be interpreted as a component-level netlist. The expansion itself is a two level function (sum of products), hence the idea to apply different radices to different levels gives the opportunity to develop a variety of two-radix combinations at the higher level of abstraction.

Green's method [28] has been chosen as an appropriate computation technique. The method is described in Chapter 2. Although it is known for its high computational complexity, it is straightforward and very flexible. At this stage of the research we are interested in the clarity rather than faster runtime. Optimisations can be applied after the entire mixed radix approach is proved valuable.

Given the motivation described in this section, the goal of the research can be defined as follows:

- Implement mixed radix synthesis and conversion approaches as a set of tools.
- Interface these new tools to the existing EDA tools, so together they form a complete design flow for secure devices.
- Explore radix combinations and analyse their efficiency with respect to the circuit parameters and security properties.

1.2 Contribution

Analysis of the existing EDA tools with respect to the security system design has been done prior to the main part of the research. The design flow structure has been proposed for conversion and synthesis approaches. Chapter 3 presents the proposed flows and discusses possible bottlenecks while interfacing between tools. The lack of tool support for encoded MVL synthesis and conversion confirmed the necessity of developing the new tools for these purposes.

Conversion driven design approach has been elaborated and implemented for binary to mixed radix circuit conversion. The basic idea behind the conversion design is

in the grouping of binary gates. Two types of gate grouping have been established: bit-wise and operandwise. Corresponding gate grouping algorithms have been developed and implemented in a tool. The tool supports netlist to netlist conversion; structural Verilog has been chosen as the input and output format.

For synthesis approach, the theory of multi-valued Reed-Muller expansions over Galois field arithmetic has been extended to mixed radix Reed-Muller expansions. The notion of a radix model has been introduced. The developed two-level radix models, presented in Chapter 4, enable computation of the following mixed radix functions:

- radix p^u functions of radix p^t arguments, where p is prime and $t > u \geqslant 1$;
- radix p^t functions of radix p arguments, where p is prime and $t > 1$;
- radix p^t functions of combined radix p and radix p^t arguments, where p is prime and $t > 1$;
- radix q functions of binary arguments, where q is any valid Galois field order.

The theory of two-level radix models and corresponding computation methods are the primary theoretical contribution of the thesis.

In order to apply the new synthesis theory, the libraries of Galois field arithmetic components have been implemented in compliance with the security requirements and covering the radices 2, 3 and 4. The implementation has been made using runtime library cells and using custom design cells.

Reed-Muller based synthesis has been implemented in RMMixed tool [1]. The tool's features include:

- computation of uniform radix and mixed radix Reed-Muller expansions in the form of a coefficient vector;
- basic logic minimisation and mapping into component level netlists;
- mapping from the component level to the gate level using provided library of components; the produced output is structural Verilog.

In addition, the tool can assist in calculating miscellaneous expressions in Galois field arithmetic. It can perform basic operations, matrix operations (including inversion), and operations on polynomials (including division). The tool is implemented in two versions: command line tool and TCL console.

The technique has been applied to a real life security related benchmarks in order to provide further motivation for its practical use. Proposed design flow has been tested down to the level of flat technology mapped netlist. Our theory allowed us to compare the efficiency of implemented component libraries in terms of energy consumption, area and delay in a range of radices and radix combinations. We have taken into account that, since the proposed theory is new, it lacks proper optimisation algorithms in comparison with well-developed binary synthesis system. Hence, the benchmark results have been analysed from the perspective of further improvement of the theory.

1.3 Organisation of the thesis

The thesis is organised as follows.

Chapter 1 (Introduction) outlines the motivation and application background of the presented research.

Chapter 2 (Background):

- provides technical background for the hardware attacks and countermeasures, outlines the prerequisites for secure design;
- gives the necessary background theory on Galois fields and Reed-Muller expansions over Galois fields. The understanding of this theoretical part is essential for reading Chapter 4.

Chapter 3 (Baseline Research) proposes the design flow stucture for conversion and synthesis approaches, gives an overview of the existing tools and outlines the requirements for the new tool. It also presents the conversion driven design

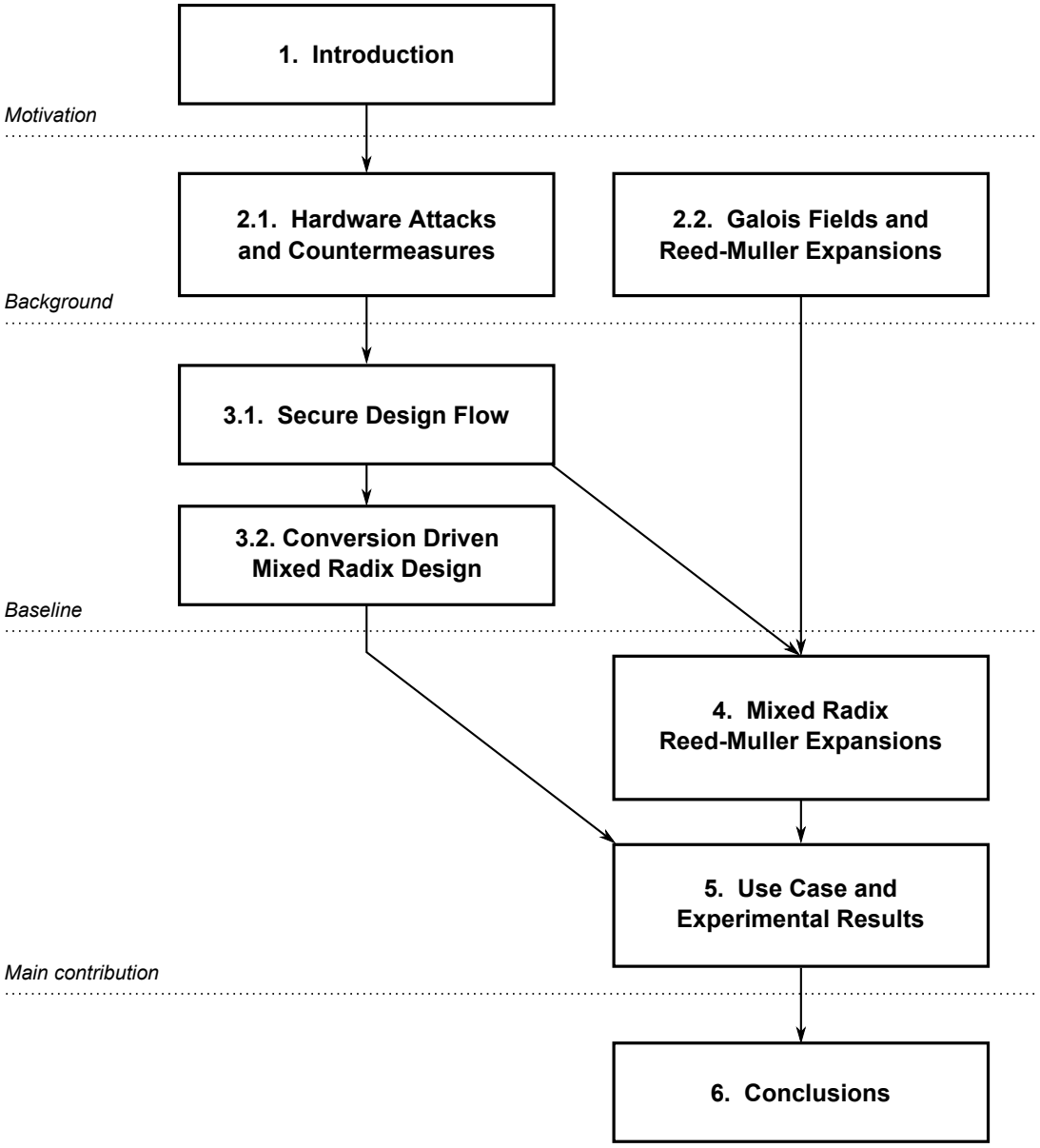


Figure 1.2: Thesis flowchart

approach – theoretical part, algorithms and benchmark results – which provides a baseline and further motivation for the main theoretical part of the thesis.

Chapter 4 (Mixed Radix Reed-Muller Expansions) presents the theory of mixed radix Reed-Muller expansions used for logic synthesis in the proposed secure design flow.

Chapter 5 (Use Case and Experimental Results) describes in details the Reed-Muller synthesis-based secure design flow, implemented tool and component libraries and discusses the benchmark results.

Chapter 6 (Conclusion) concludes the work and outlines the possible future development of the research.

Appendix A (RMMixed User Manual) contains the user manual for Reed-Muller synthesis tool.

Some chapters present independent approaches, so there is a certain concurrency in the structure of the thesis. Figure 1.2 illustrates the flow.

Chapter 2

Background

The presented research connects two independent aspects: hardware level security and the theory of multi-valued Reed-Muller expansions. This chapter covers the background information for both.

2.1 Hardware Attacks and Countermeasures

As has been discussed in Chapter 1, in order to make the device sufficiently protected against the hardware attacks, certain guidelines must be followed. This section describes the possible dangers and gives the basic understanding of the protection at the hardware level using the special type of signal encoding, known as switching balanced codes.

2.1.1 Types of attacks and countermeasures

Cryptanalysis is mainly divided into software attacks and hardware attacks. Figure 2.1 shows different types of attacks grouped by the field of action [60].

Software cryptanalysis is targetted directly at the cryptographic algorithms, acting at the mathematical level or the level of programming language. This type of attack is not addressed in thie presented research.

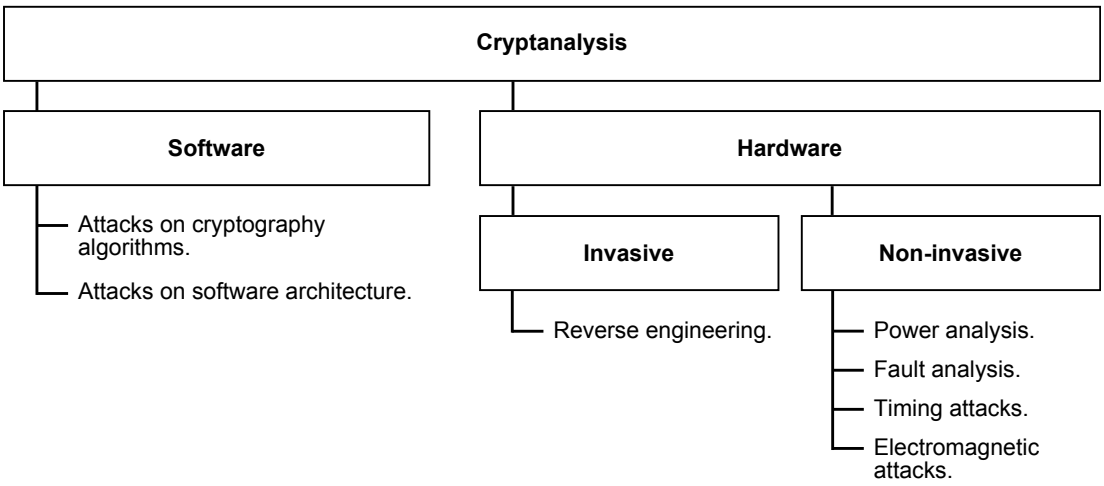


Figure 2.1: Types of cryptanalysis

Hardware attacks behave indirectly, using the imperfections of the actual implementations rather than those of the algorithms. These attacks are divided into two major categories: invasive and non-invasive (side-channel). Based on reverse engineering, invasive attacks require special laboratory equipment and destroy packaging in the process while side-channel attacks do not require in-depth knowledge of the technology and use simple equipment [60]. With respect to the physical parameter used as the source of information, side-channel attacks are known as power analysis, timing analysis, electromagnetic analysis and fault analysis.

During the *power analysis* the hacker monitors data-dependent power consumption of the device during the normal operation [39]. The method relies on the following fundamental hypothesis: there exists an intermediate variable that appears during the computation of the algorithm, such that knowing a few key bits allows the attacker to decide whether two inputs (respectively two outputs) do or do not give the same value for this variable [18]. Consequently, splitting such variables and combining them with random values can protect against power analysis. This method is called *masking*, and its major advantage with respect to this work is that it can be implemented using standard EDA software. However, recent research has discovered a mathematical modification of power analysis that can break the masking approach [48].

Another countermeasure to the power analysis is to make the device's power consumption data-independent, it is called *power balancing*. A number of methods to equalise the power signatures using specific representation of data signals over physical wires, e.g. m-of-n codes, were proposed in [17, 46, 67, 43]. M-of-n codes are an encoding scheme in which data is represented using n wires and where m of them are set to an active level (usually high). A protocol separating data using dummy symbols (spacers) is called a *spacer protocol*. In this configuration, the number of wires that switch during the cycle is constant, so is the power consumption. A particular emphasis has so far only been put on dual-rail (1-of-2) codes for binary radix and 1-of-4 for quaternary logic. As a price for the improved resistance to power attacks the approach results in an overhead with respect to the overall power consumption of the system. More details on power balancing using m-of-n codes is given in Subsection 2.1.2.

Electromagnetic analysis is similar to power analysis but uses data-dependent electromagnetic emission instead of power consumption [26]. The method of equalising switching activity of the logic can work in this case as well.

Another way to make the hackers' life harder is to randomise the timing properties of the circuits, so it would become difficult to perform correct sampling of physical parameters during operation. The simplest method is to insert random delays in the clock cycles. The most reliable countermeasure however is an asynchronous logic design (i.e. self-timed circuits) [46]. Asynchronously working device modules cause overlays in power consumption, thus making it practically impossible to distinguish between single operations. This can help against the *timing analysis*, which uses the amount of time required for running non constant cryptographic algorithm to retrieve information about the data processed [41].

Fault analysis is an attack which uses abnormal environment conditions, e.g. glitches on power or clock signals, so malfunction of the device can create a window for vulnerabilities [15]. A known countermeasure for this attack is using fault-tolerant protocols. M-of-n codes are fault-tolerant as they imply relatively simple fault detection

logic [14].

Since m-of-n codes and clockless design approach appear to be the most universal countermeasures, the following discussion on the secure design is presented in relation to these ideas.

2.1.2 Power balancing using m-of-n codes

Power balancing is one of the countermeasures to power analysis. It's concept is to create data-independent power consumption using so-called switching balanced codes and power balanced implementations of logic gates.

CMOS power

Digital circuits consume power whenever they perform computation. Using a constant power supply and input signals to execute, logic cells draw current from the supply and dissipate energy as heat. Power consumption of the circuit is a sum of the power consumption of the individual logic cells.

The overall power dissipation P_{total} of a static CMOS circuit is composed of three components: switching power P_{switch}, short-circuit power P_{short} and leakage power P_{leak};

$$P_{total} = P_{switch} + P_{short} + P_{leak}.$$

Short-circuit power P_{short} is dissipating due to the presence of a direct path between supply and ground during logical transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$). Leakage power dissipation P_{leak}, known as static power consumption, comes from different sources: sub-threshold voltage, tunneling through gate oxide, and through reverse biased diodes. Although at the submicron level the leakage power becomes a relevant factor, short-circuit power and leakage power are considered relatively small compared to the switching power [47].

Switching power P_{switch}, also called dynamic power or dynamic dissipation, is attributed to the charging and discharging of capacitances: gate outputs (load capacit-

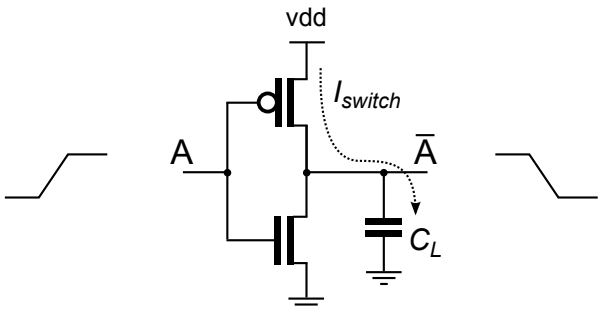


Figure 2.2: Switching activity in CMOS inverter

ances) and transistor-to-transistor connections inside gates (intrinsic capacitances) [38];

$$P_{switch} = P_{load} + P_{internal}.$$

An inverter, shown in Figure 2.2, does not have intrinsic capacitances, hence the load power is equivalent to the dynamic power. During a $0 \rightarrow 1$ transition, current flows from the supply charging the load capacitance C_L. Half of the energy is stored in the load capacitor, and the other half is dissipated by the pull-up PMOS network. During a discharging transition $1 \rightarrow 0$ current flows from the load to ground, and the energy stored in the load capacitor is dissipated by the pull-down NMOS network; no energy is drawn from the supply. Each switching cycle takes a fixed amount of energy equal to $C_L \cdot V_{dd}^2$. The inverter's average dynamic power consumption over some time period is the integral of the instantaneous power multiplied by switching frequency f of $0 \rightarrow 1$ transitions.

Other CMOS gates and networks of interconnected gates display switching properties other than those of a inverter. Furthermore, they exhibit "lazy switching" characteristics causing a memory effect, where the logical values of outputs are retained from the earlier computation cycles. This memory effect occurs when a gate output computes to the same logical value as the existing output; this naturally happens when more than one set of inputs maps to the same outputs. Similarly, internal nodes exhibit such memory effects and will discharge or charge deterministically. For example, consider the 2-input NAND gate shown in Figure 2.3. It has two parallel

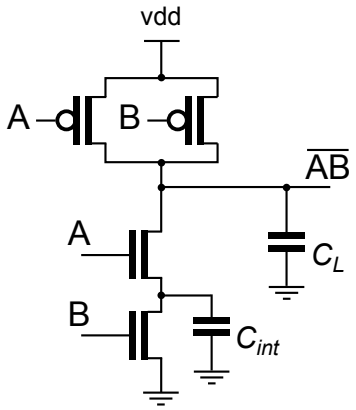


Figure 2.3: CMOS NAND gate

PMOS transistors and two NMOS transistors connected in series forming the internal transistor-to-transistor capacitance C_{int}. Assume the inputs are either 00 or 01 and C_{int} is discharged; if in the next cycle 00 or 01 arrives, then the output will retain its existing value and no power will be consumed. If in the next cycle 10 arrives, the output will retain its value, but C_{int} will charge, meaning it can freely discharge at any point in subsequent cycles.

Estimating the power consumption of complex gates and networks requires taking into account switching statistics. For instance, from the truth table of the NAND gate the probability of $0 \rightarrow 1$ transition is 3/16, the probability of $1 \rightarrow 0$ transition is also 3/16, and the probability that the output will be retained is 10/16. The concept of switching activity is used to determine the probability of the output switching. For N periods of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions, the switching activity α determines how many $0 \rightarrow 1$ transitions occur. Thus,

$$P_{load} = \alpha C_L \cdot f \cdot V_{dd}^2,$$

where α is the switching activity, C_L is output capacitance, and f is the switching frequency of $0 \rightarrow 1$ transitions. For the inverter, $\alpha = 1$ and $P_{load} = C_L \cdot f \cdot V_{dd}^2$.

$$P_{internal} = \sum_{i=1}^{n} \alpha_i \cdot C_i \cdot f \cdot V_i \cdot V_{dd},$$

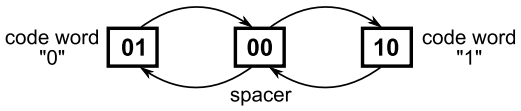


Figure 2.4: Dual-rail (1-of-2) protocol

estimated using switching activity α_i, parasitic capacitance C_i and internal voltage V_i for each node i. However, in most cases C_L is considered to be much bigger than the sum of C_i, so the internal power consumption is often neglected, and the switching power of a particular cell is determined by its load power P_{load}.

m-of-n codes

Since the main source of power consumption in the CMOS netlist is the switching activity of gates, the method to equalise it is to produce a symmetric switching. This can be achieved using switching balanced codes, for example m-of-n codes.

m-of-n codes are an encoding scheme in which data is represented using n wires and where m of them are set to an active level (usually high).

The simplest example is 1-of-2 code, called dual-rail, which uses a pair of wires per bit of information d; one wire d_1 is used for signaling a logic 1 (or true), and another wire d_0 is used for signaling logic 0 (or false). Traditionally this code is employed to represent data in self-timed circuits [21]. Viewed together the d_1d_0 wire pair is a codeword; $d_1d_0 = 10$ and $d_1d_0 = 01$ represent "valid data" (logic 0 and logic 1 respectively) and $d_1d_0 = 00$ represents "no data" ("spacer" or NULL). The codeword $d_1d_0 = 11$ is not used, and a transition between valid codewords must be done via the spacer, as illustrated in Figure 2.4.

In asynchronous circuits, dual-rail protocol is used to create a 4-phase handshake protocol, which replaces the clock signal. An abstract view of 4-phase handshaking consist of (1) the sender issues a valid codeword, (2) the receiver absorbs the codeword and sets acknowledge high, (3) the sender responds by issuing the empty codeword, and (4) the receiver acknowledges this by taking acknowledge low [68]. Dual-rail handshake circuit is shown in Figure 2.5.

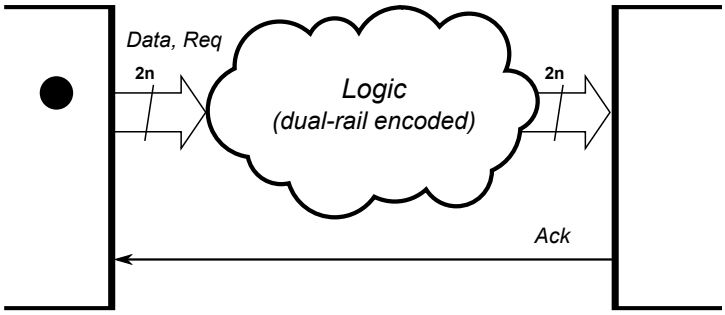


Figure 2.5: 4-phase handshake

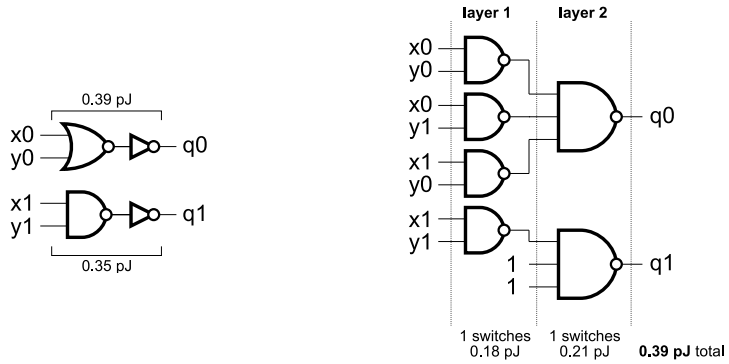
Table 2.1: Encoded binary, ternary and quaternary signals.

radix	value	encoded signal
1-of-2 (dual-rail)		
binary	0	01
	1	10
	NULL	00
1-of-3		
ternary	0	001
	1	010
	2	100
	NULL	000
1-of-4		
quaternary	0	0001
	1	0010
	2 or A	0100
	3 or B	1000
	NULL	0000

An important feature of the dual-rail encoding is its balanced power consumption which facilitates circuit resistance to power analysis attacks. In particular, switching from a spacer to any code word consumes the same amount of power due to the symmetry between rails. The same is true for all m-of-n generalised encodings. Apart from dual-rail we use 1-of-4 to represent quaternary signals and 1-of-3 for ternary. Table 2.1 shows how these encodings can be applied.

single-rail			dual-rail		
x	y	$q = xy$	$x = (x_1, x_0)$	$y = (y_1, y_0)$	$q = (q_1, q_0)$
0	0	0	01	01	01
0	1	0	01	10	01
1	0	0	10	01	01
1	1	1	10	10	10
–	–	–	00	XX	00
–	–	–	XX	00	00

Table 2.2: Dual-rail AND gate truth table and its single-rail equivalence



(a) relaxed balancing

(b) fully balanced

Figure 2.6: RTL implementions of dual-rail encoded AND gate showing different levels of power balancing.

Cell implementations for balancing

Since the primary attribute of these codes is balanced switching, the components should also display this feature. Ideally the form and size of the power signature of a component should be symmetrical with respect to switching from spacer to data and vice versa. Usually this is achieved by introducing additional dummy-logic paths. For real life examples an ideal symmetry is impossible, but the components can have certain maximum balancing allowed by the technology capabilities. Such implementations are henceforth called *fully balanced*. In contrast, the notion of *relaxed balancing* is used when the security is slightly compromised for significant power and area gains

To demonstrate the difference between fully balanced components and components

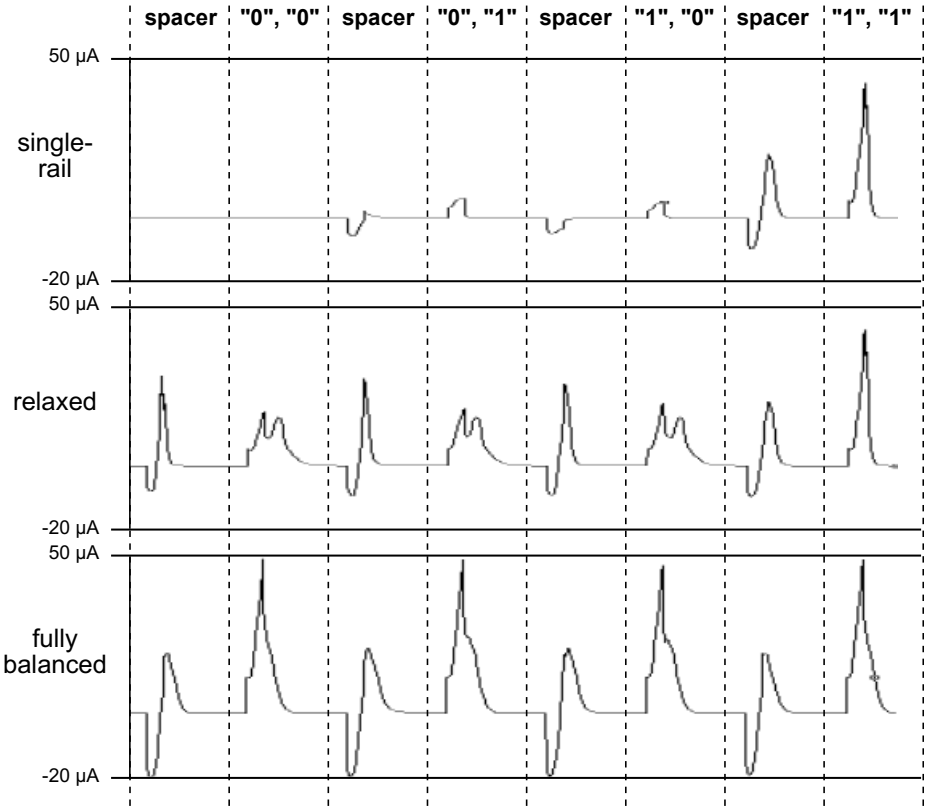
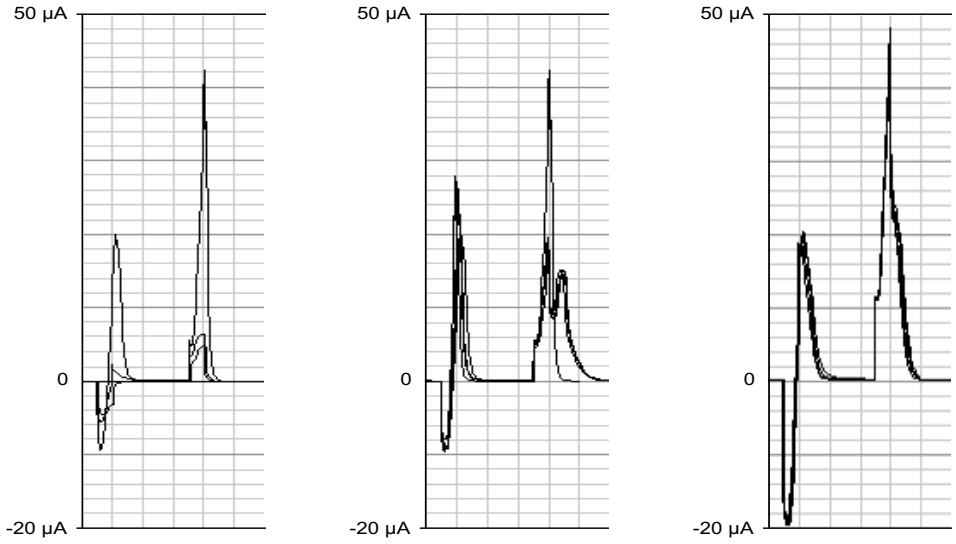


Figure 2.7: Power signatures for different data transitions illustrating the imbalance.



(a) single-rail (b) relaxed balancing (c) fully balanced

Figure 2.8: Power signatures (overlayed)

with so-called "relaxed" balancing, consider a binary AND gate. Table 2.2 shows its truth table. A straightforward dual-rail implementation is shown in Figure 2.6(a). Switching dual-rail inputs $[x = (x_1, x_0), y = (y_1, y_0)]$ from the spacer value to ["0", "0"], ["1", "0"] and ["0", "1"] causes the NOR gate to fire. Switching from the spacer to ["1", "1"] fires the NAND gate. NAND and NOR gates have different switching energy values thus in this case the component is weakly balanced.

In order to improve the balance we have to add logic paths that make the structure of the component symmetrical with respect to the switching activity of the gates and input signals; this is shown in Figure 2.6(b). In the spacer state all inputs are set to low, thus all the outputs of the 2-input NAND gates in the first layer are set to high, precharging the NAND gates in the second layer. Arrival of a data signal (["0", "0"], ["0", "1"], ["1", "0"], or ["1", "1"]) causes exactly one gate from the first layer to fire. This will produce only one 0 signal to the second layer, switching one of 3-input NANDs. Addition of constant inputs to one of the gates guarantees that all gates in each layer are the same.

Although there are certain unavoidable aspects of the technology, such as transistor level asymmetry which introduces some imbalance even to this design, any implementation is acceptable if it fits the requirements of the security standard [5]. For the same reason, the structure shown in Figure 2.6(a) might also be sufficient, since the difference in switching energies is not large. Figure 2.7 displays the analogue simulation results for the components shown in Figure 2.6 along with the non-encoded single-rail implementation. As discussed above, fully balanced implementation provides maximum security, while non-encoded circuit is totally unprotected. This is a typical illustration why the standard design flow and tools cannot be directly applied in this area.

A detailed discussion on secure implementations and power balancing can also be found in [31].

2.2 Galois Fields and Reed-Muller Expansions

The section gives a brief yet sufficient background on Galois fields arithmetic, multi-valued Reed-Muller expansions over Galois fields and Green's direct method of computation. The understanding of these theoretical aspects is essential for understanding the theory of mixed radix Reed-Muller expansions. All the theorems in this section are given without proofs. The proofs can be found in the referenced literature. For the extended study of the Galois theory, the reader may refer to [30, 62, 66].

2.2.1 Galois Fields

Rings

Definition 2.1. Let R be a nonempty set on which we have two closed binary operations, denoted by $+$ and $\cdot$, i.e. $a + b \in R$ and $a \cdot b \in R$ for all $a, b \in R$. Then $(R, +, \cdot)$ is a *ring* if for all $a, b, c, \in R$ the following conditions are satisfied [30]:

1. Commutative law of $+$: $a + b = b + a$
2. Associative law of $+$: $a + (b + c) = (a + b) + c$
3. There exists a *zero element* [66] (also called *additive identity* [30]), denoted as $0 \in R$, such that $a + 0 = 0 + a = a$ for every a in R.
4. For each $a \in R$ there is an element $b \in R$ with $a + b = b + a = 0$. This element is called the *additive inverse* of a and usually denoted as $-a$.
5. Associative law of $\cdot$: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
6. Distributive law of $\cdot$ over $+$:

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

$$(b + c) \cdot a = b \cdot a + c \cdot a$$

If, in addition, $a \cdot b = b \cdot a$ for all $a, b \in R$ then R is called a *commutative ring*.

A *ring with unity* is a ring R containing an element, denoted by 1 and called the *unity* or the *multiplicative identity*, such that for all $a \in R$

$$1 \cdot a = a \cdot 1 = a$$

Definition 2.2. An *integral domain* is a commutative ring with unity R such that the equation $ab = 0 \in R$ implies that either $a = 0$ or $b = 0$ [66].

Theorem 2.1. *A commutative ring with unity* R *is an integral domain if and only if it satisfies the* cancellation law*: if* $ra = rb$ *and* $r \neq 0$*, then* $a = b$ *[62].*

Definition 2.3. Let R be a ring with unity. If $a \in R$ and there exists $b \in R$ such that $ab = ba = 1$, then b is called a *multiplicative inverse* of a (usually denoted as a^{-1}) and a is called a *unit* of R. The element b is also a unit of R [30].

Definition 2.4. Commutative ring with unity R is called a *field* if every nonzero element of R is a unit [30].

$\mathbb{Z}_n$ rings

The presented research uses *finite fields*, i.e. fields with finite number of elements. The simpliest example is $\mathbb{Z}_n$ rings.

For a fixed positive integer n, define the ring $\mathbb{Z}_n$ of integers modulo n as follows [62]. Its elements are the subsets of $\mathbb{Z}$

$$\begin{aligned} [a] &= \{m \in \mathbb{Z} : m \equiv a \pmod{n}\} \\ &= \{m \in \mathbb{Z} : m = a + kn \text{ for some } k \in \mathbb{Z}\} \end{aligned}$$

where $a \in \mathbb{Z}$ ($[a]$ is called the congruence class of a mod n). Addition and multiplication are given by

$$[a] + [b] = [a + b]$$

$$[a]\,[b] = [ab]$$

and $[1]$ is unity. It is routine to check that addition and miltiplication are well defined (that is, if $a \equiv a' \pmod{n}$ and $b \equiv b' \pmod{n}$, then $a + b \equiv a' + b' \pmod{n}$ and $ab \equiv a'b' \pmod{n}$; i.e, $[a] + [b] = [a'] + [b']$ and $[a]\,[b] = [a']\,[b']$), and that $\mathbb{Z}_n$ is a ring under these operations [62].

It is a common practice, when working with $\mathbb{Z}_n$, to eliminate brackets from the notation. In $\mathbb{Z}_3$, for example, it is correct to write $2 + 2 = 1$.

Theorem 2.2. *$\mathbb{Z}_n$ is an integral domain if and only if n is prime [62].*

Theorem 2.3. *If p is a prime, then $\mathbb{Z}_p$ is a field [62].*

Polynomial rings over fields

Definition 2.5. Let R be a ring. The ring *of polynomials in the transcendental (indeterminate* [30]*) variable x, with the coefficients in* R, is denoted by $R[x]$ and consists of the set of all formal expressions of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

with each $a_i \in R$ [66]. The element $a_t \in R$ is called the coefficient of x^t in $f(x)$ and two polynomials are considered to be equal if and only if, for each t, their nonzero coefficients of x^t are equal. The *zero polynomial* has every coefficient equal to zero and is abbreviated to 0.

The *degree* of $f(x)$ is the largest n such that the coefficient of x^n is nonzero and is denoted by $\deg(f(x))$. Hence $\deg(0) = 0$, $\deg(a_0) = 0$ and $\deg(ax + b) = 1$.

If $g(x) = b_m x^m + b_{m-1} x^{m-1} + \ldots + b_0$ we define the *sum*, $f(x) + g(x)$, to be the polynomial whose coefficient of x^t is $a_t + b_t$, and we define the *product*, $f(x)\,g(x)$, to be the polynomial whose coefficient of x^t is

$$a_t b_0 + a_{t-1} b_1 + \ldots + a_1 b_{t-1} + a_0 b_t$$

Corollary 1. *Let* $R[x]$ *be a polynomial ring.*

1. *If* R *is commutative, then* $R[x]$ *is commutative.*

2. *If* R *is a ring with unity, then* $R[x]$ *is a ring with unity.*

3. $R[x]$ *is an integral domain if and only if* R *is an integral domain [30].*

Irreducible polymonials: finite fields

We now wish to construct finite fields other than those of type $(\mathbb{Z}_p, +, \cdot)$, where p is a prime. The construction will use the following special polynomials.

Definition 2.6. Let $f(x) \in F[x]$, with F a field and $\deg(f(x)) \geqslant 2$. We call $f(x)$ *reducible* (over F) if there exists $g(x), h(x) \in F[x]$, where $f(x) = g(x)\,h(x)$and each of $g(x)$, $h(x)$ has degree $\geqslant 1$. If $f(x)$ is not reducible it is called *irreducible*, or *prime* [30].

Theorem 2.4. *Let* $s(x) \in F[x]$, $s(x) \neq 0$. *Define relation* $\mathcal{R}$ *on* $F[x]$ *by* $f(x)\,\mathcal{R}\,g(x)$ *if* $f(x) - g(x) = t(x)\,s(x)$, *for some* $t(x) \in F[x]$ — *that is,* $s(x)$ *divides* $f(x) - g(x)$ *[30]. Then* $\mathcal{R}$ *is an equivalence relation on* $F[x]$.

When the situation described in Theorem 2.4 occurs, we say that $f(x)$ is *congruent* to $g(x)$ *modulo* $s(x)$ and write $f(x) \equiv g(x) \pmod{s(x)}$. The relation $\mathcal{R}$ is referred as *congruence modulo* $s(x)$.

Let us examine the equivalence classes of one such relation. Let $s(x) = x^2 + x + 1 \in$

$\mathbb{Z}_2[x]$. Then

$$\begin{aligned}
[0] &= \left[x^2+x+1\right] = \left\{0, x^2+x+1, x\left(x^2+x+1\right), (x+1)\left(x^2+x+1\right), \ldots\right\} \\
&= \left\{t(x)\left(x^2+x+1\right) | t(x) \in \mathbb{Z}_2[x]\right\} \\
[1] &= \left\{1, x^2+x, x\left(x^2+x+1\right)+1, (x+1)\left(x^2+x+1\right)+1, \ldots\right\} \\
&= \left\{t(x)\left(x^2+x+1\right)+1 | t(x) \in \mathbb{Z}_2[x]\right\} \\
[x] &= \left\{x, x^2+1, x\left(x^2+x+1\right)+x, (x+1)\left(x^2+x+1\right)+x, \ldots\right\} \\
&= \left\{t(x)\left(x^2+x+1\right)+x | t(x) \in \mathbb{Z}_2[x]\right\} \\
[x+1] &= \left\{x+1, x^2, x\left(x^2+x+1\right)+(x+1), (x+1)\left(x^2+x+1\right)+(x+1), \ldots\right\} \\
&= \left\{t(x)\left(x^2+x+1\right)+(x+1) | t(x) \in \mathbb{Z}_2[x]\right\}
\end{aligned}$$

We now place a ring structure on these equivalence classes [30]. Recalling how this was accomplished in Subsection 2.2.1 for $\mathbb{Z}_n$, we define addition by $[f(x)] + [g(x)] = [f(x) + g(x)]$. Since $\deg(f(x)+g(x)) \leqslant \max\{\deg(f(x)), \deg(g(x))\}$, we can always find the equivalence class for $[f(x)+g(x)]$. Here, for example, $[x]+[x+1] = [x+(x+1)] = [2x+1] = [1]$ because $2 = 0$ in $\mathbb{Z}_2$.

Defining the multiplication of these equivalence classes is a bit more tricky. For instance, what is $[x][x]$? If, in general, we define $[f(x)][g(x)] = [f(x)g(x)]$, it is possible that $\deg(f(x)g(x)) \geqslant \deg(s(x))$, so we may not readily find $[f(x)g(x)]$ in the list of equivalence classes. However, if $\deg(f(x)g(x)) \geqslant \deg(s(x))$, then using the division algorithm, we can write $f(x)g(x) = q(x)s(x) + r(x)$, where $r(x) = 0$ or $\deg(r(x)) < \deg(s(x))$. With $f(x)g(x) = q(x)s(x) + r(x)$ it follows that $f(x)g(x) \equiv r(x) \pmod{s(x)}$, and we define $[f(x)g(x)] = [r(x)]$, where $r(x)$ does occur in the list of equivalence classes.

From these observations we construct tables, shown in Figure 2.9, for the addition and multiplication of $\{[0], [1], [x], [x+1]\}$. In these tables we write a for $[a]$.

Theorem 2.5. *Let* $s(x)$ *be a nonzero polynomial in* $F[x]$.

1. *The equivalence classes of* $F[x]$ *for the relation of congruence modulo* $s(x)$ *form a*

$+$	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

$\cdot$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Figure 2.9: Addition and multiplication of $\{[0], [1], [x], [x+1]\}$

commutative ring with unity under the closed binary operations

$$[f(x) + g(x)] = [f(x)] + [g(x)]$$

$$[f(x)][g(x)] = [f(x)\, g(x)] = [r(x)]$$

where $r(x)$ *is the remainder obtained upon dividing* $f(x)\, g(x)$ *by* $s(x)$*. This ring is denoted by* $F[x] / (s(x))$*.*

2. *If* $s(x)$ *is irreducible in* $F[x]$*, then* $F[x] / (s(x))$ *is a field.*

3. *If* $|F| = q$ *and* $\deg(s(x)) = n$*, then* $F[x] / (s(x))$ *contains* q^n *elements [30].*

Definition 2.7. Let $(R, +, \cdot)$ be a ring. If there is a least positive integer n such that $nr = r + \ldots + r\,(n\text{times}) = 0 \in R$ for all $r \in R$, then we say that R has *characteristic* n and write $\text{char}(R) = n$. When no such integer exists, R said to have characteristic 0.

Theorem 2.6. *Let* $(F, +, \cdot)$ *be a field. If* $\text{char}(F) > 0$*, then* $\text{char}(F)$ *must be prime.*

Theorem 2.7. *A finite field* F *has order* p^t*, where* p *is a prime and* $t \in \mathbb{Z}^+$*.*

Theorem 2.8. *(Galois) For every prime* p *and every positive integer* t*, there exists a field having exactly* p^t *elements [62].*

Corollary 2. *(E.H. Moore) Any two finite fields of order* p^t *are isomorphic [62].*

These fields were discovered by the French mathematician Evariste Galois (1811–1832) in his work on the nonexistence of formulas for solving general polynomial equations of degree $\geqslant 5$ over $\mathbb{Q}$. As a result, a finite field of order p^t is denoted by $\text{GF}(p^t)$, where the letters GF stand for *Galois field*.

Definition 2.8. If E is a field and F is a subset which, under the operations of E, is itself a field then F is called a *subfield* of E and E is an *extension* of F [66]. GF(p) is a subfield of GF(p^t) for any prime p and $t > 1$.

Definition 2.9. A *homomorphism* $\psi : F \to F'$ *of fields* is a ring map

$$(\psi(a+b) = \psi(a) + \psi(b), \psi(ab) = \psi(a)\psi(b))$$

such that $\psi(1) = 1$. If ψ is one-to-one and onto, it is an *isomorphism* (an *automorphism* if $F = F'$). [66].

If the fields F and F' are homomorphic only with respect to one of the operations, i.e. either $\psi(a+b) = \psi(a) + \psi(b)$ or $\psi(ab) = \psi(a)\psi(b)$ is true for $\psi : F \to F'$, the field F is called a *subgroup* of F' under the operation of addition (multiplication).

The next subsection summarises the properties of Galois fields used in the definition and the computation of Reed-Muller expansions.

Galois field properties overview

For any $q = p^t$, where p is a prime number and integer $t \geqslant 1$, a Galois field GF(q) is defined as an algebraic structure consisting of:

1. q elements;
2. closed binary operations $+$ and $\cdot$ called addition and multiplication respectively, i.e. $a + b \in GF(q)$ and $a \cdot b \in GF(q)$ for all $a, b \in GF(q)$;
3. elements 0 and 1 such that $a + 0 = a$ and $a \cdot 1 = a$ for all $x \in GF(q)$.
4. Operations of addition and multiplication have the property of commutativity, transitivity, and the operation of multiplication is distributive over addition.

For every $a \in GF(q)$, there exists $-a \in GF(q)$ such that $a + (-a) = 0$ [52]. Similarly, for every nonzero $b \in GF(q)$, there exists $b^{-1} \in GF(q)$ such that $b \cdot b^{-1} = 1$.

GF(2):

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

GF(3):

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

GF(4):

+	0	1	A	B
0	0	1	A	B
1	1	0	B	A
A	A	B	0	1
B	B	A	1	0

×	0	1	A	B
0	0	0	0	0
1	0	1	A	B
A	0	A	B	1
B	0	B	1	A

Figure 2.10: Addition and multiplication over some finite fields

Another important property of the Galois fields is that for every $a \in GF(q)$, $a^q = a$ and for $a \neq 0$, $a^{q-1} = 1$ [52]. Hence the number of possible powers of a in GF(q) is limited to q.

The elements of prime Galois fields are denoted as integer numbers. In the case of GF(p^t), first p elements are denoted as integers, as they are the elements of the prime subfield GF(p), and the rest of the elements assign consequent uppercase latin letters: $A, B, \ldots$. For example the elements of GF(4) are: $0, 1, A, B$.

Arithmetic operations for some Galois fields used in the further examples are shown in Figure 2.10.

Matrix operations over Galois fields

The presented research extensively uses matrix operations over Galois fields. Due to the properties of Galois fields, the matrix operations are defined in the same way as in regular algebra. The elements of matrices are the elements of some field GF(q).

Matrix addition is defined as

$$[X + Y]_{i,j} = X_{i,j} + Y_{i,j},$$

where X, Y are m-by-n matrices, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$.

If X is an m-by-n matrix and Y is n-by-p matrix, then their *matrix product* XY is the m-by-p matrix whose entries are given by dot-product of the corresponding row of X and the corresponding column of Y:

$$[XY]_{i,j} = \sum_{r=1}^{n} X_{i,r} Y_{r,j},$$

where $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant p$. Matrix multiplication satisfies the rules $(XY)Z = X(YZ)$ (associativity), and $(X+Y)Z = XZ+YZ$ as well as $Z(X+Y) = ZX+ZY$ (left and right distributivity), whenever the size of the matrices X, Y, Z is such that the various products are defined.

In linear algebra an n-by-n (square) matrix X is called *invertible*, if there exists an n-by-n matrix Y such that

$$XY = YX = I_n$$

where I_n denotes the n-by-n identity matrix and the multiplication used is ordinary matrix multiplication. If this is the case, then the matrix Y is uniquely determined by X and is called the *inverse* of X, denoted by X^{-1}. Non-square matrices (m-by-n matrices for which $m \neq n$) do not have an inverse.

Definition 2.10. If X is an m-by-n matrix and Y is a p-by-q matrix, then the *Kronecker product* [23] is the mp-by-nq block matrix

$$X \otimes Y = \begin{bmatrix} x_{1,1}Y & \cdots & x_{1,n}Y \\ \vdots & \ddots & \vdots \\ x_{m,1}Y & \cdots & x_{m,n}Y \end{bmatrix}$$

Kronecker product is not commutative, but it is associative: $(X \otimes Y) \otimes Z = X \otimes (Y \otimes Z)$ for matrices X, Y, Z. One of the properties of Kronecker product used in the presented theory is that for any invertible matrices X and Y, $(X \otimes Y)^{-1} = X^{-1} \otimes Y^{-1}$; $(X \otimes Y)$ is invertible only if X and Y are invertible.

2.2.2 Multi-valued Reed-Muller expansions

Binary and multi-valued functions can be represented using XOR sum of products. One particular case is Reed-Muller (RM) expansions. In the multi-valued case, Reed-Muller expansions have the form of sum of products computed over GF(q).

The following simplified definition of literal is used in this thesis; it is sufficient for understanding the theory at the required level. For classic definition of literal and literal form of Reed-Muller expansions, refer to [35, 34].

Definition 2.11. *Literal* $\widetilde{x}$ of the q-valued variable x is one of q possible polarity forms $(x + c)$; c is an element of GF(q) denoting the literal.

Definition 2.12. For an n-variable q-valued function *polarity number* k is defined as an integer representation of a q-nary tuple $\langle k_n \dots k_1 \rangle_q = k$ where $k_i \in \mathrm{GF}(q)$ denotes the polarity of literal $\widetilde{x}_i$, $i = 1 \dots n$. For example, the polarity number $k = 2 = \langle 0A \rangle_4$ for 2-variable quaternary function means that $\widetilde{x}_2 = x_2$ and $\widetilde{x}_1 = x_1 + A$.

Definition 2.13. A *general canonical Reed-Muller (RM) expansion* for an n-variable q-valued function in polarity k is defined as follows:

$$f(\widetilde{x}_1, \dots, \widetilde{x}_n) = \sum_{i=0}^{q^n-1} a_i \left[\prod_{j=1}^{n} \widetilde{x}_j^{i_j} \right] \quad \text{over GF}(q) \tag{2.1}$$

where i_j is a single digit of a q-nary tuple $\langle i_n \dots i_1 \rangle_q = i$. Vector $a = \begin{bmatrix} a_0 & \dots & a_{q^n-1} \end{bmatrix}^t$ is a *coefficient vector*. In a *fixed polarity* RM expansion each variable must be represented by the same literal throughout the expansion.

Following from Definition 2.10 of the Kronecker product, (2.1) can be rewritten in matrix form. Let $\widetilde{X}_j = \begin{bmatrix} 1 & \widetilde{x}_j & \dots & \widetilde{x}_j^{q-1} \end{bmatrix}$ is the vector of all possible powers of the literal $\widetilde{x}_j$, then

$$f(\widetilde{x}_1, \dots, \widetilde{x}_n) = \left(\widetilde{X}_n \otimes \dots \otimes \widetilde{X}_1 \right) \cdot a \quad \text{over GF}(q) \tag{2.2}$$

where a is a coefficient vector as in Definition 2.13. Equation (2.2), as well as (2.1), represents the sum of all possible products of all possible powers of the input variables

multiplied by coefficients from a.

Reed-Muller expansion of zero polarity for a quaternary function of one variable takes the form:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \quad \text{over GF(4)} \tag{2.3}$$

As an example, a 2-variable quaternary function of the polarity $k = 2 = \langle 0A \rangle_4$ gives the following RM form:

$$\begin{aligned} f(x_1, x_2) \;=\; & a_0 + a_1 \ddot{x}_1 + a_2 \ddot{x}_1^2 + a_3 \ddot{x}_1^3 + \\ & a_4 x_2 + a_5 \ddot{x}_1 x_2 + a_6 \ddot{x}_1^2 x_2 + a_7 \ddot{x}_1^3 x_2 + \\ & a_8 x_2^2 + a_9 \ddot{x}_1 x_2^2 + a_{10} \ddot{x}_1^2 x_2^2 + a_{11} \ddot{x}_1^3 x_2^2 + \\ & a_{12} x_2^3 + a_{13} \ddot{x}_1 x_2^3 + a_{14} \ddot{x}_1^2 x_2^3 + a_{15} \ddot{x}_1^3 x_2^3 \end{aligned}$$

computed over GF(4); $\ddot{x}_1 = x_1 + A$.

2.2.3 Green's direct method

The computation of an RM expansion corresponds directly to the computation of the coefficient vector a. The following method of computation has been proposed in [28] for quaternary RM expansions and is known as *Green's direct method*.

Let $d = \begin{bmatrix} d_0 & \dots & d_3 \end{bmatrix}^t$, $d_0 \dots d_3 \in \text{GF}(4)$ be the truth vector of the zero polarity quaternary function $f(x)$ shown in (2.3). Then,

$$\begin{aligned} d_0 &= f(0) = a_0 \\ d_1 &= f(1) = a_0 + a_1 + a_2 + a_3 \\ d_2 &= f(A) = a_0 + A a_1 + B a_2 + a_3 \\ d_3 &= f(B) = a_0 + B a_1 + A a_2 + a_3 \end{aligned} \tag{2.4}$$

This set of linear equations can be solved in matrix form:

$$d = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & A & B & 1 \\ 1 & B & A & 1 \end{bmatrix} \cdot a = S_0 a \quad \text{over GF(4)} \tag{2.5}$$

$$a = S_0^{-1} d = W_0 d \quad \text{over GF(4)} \tag{2.6}$$

$$W_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & B & A \\ 0 & 1 & A & B \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Hence a coefficient vector a can be derived from the truth vector of a function. For non-zero polarity forms ($k = 1 \ldots 3$) of a 1-variable quaternary function the same approach can be applied, and corresponding matrices W_1, W_2, W_3 can be found:

$$S_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & B & A & 1 \\ 1 & A & B & 1 \end{bmatrix}, \quad W_1 = S_1^{-1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & A & B \\ 1 & 0 & B & A \\ 1 & 1 & 1 & 1 \end{bmatrix};$$

$$S_2 = \begin{bmatrix} 1 & A & B & 1 \\ 1 & B & A & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad W_2 = S_2^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ B & A & 0 & 1 \\ A & B & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix};$$

$$S_3 = \begin{bmatrix} 1 & B & A & 1 \\ 1 & A & B & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad W_3 = S_3^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ A & B & 1 & 0 \\ B & A & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Generally, for any n-variable quaternary function defined by its truth vector d, a coefficient vector a of k-polarity RM expansion can be found from the following equation:

$$d = (S_{k_n} \otimes \ldots \otimes S_{k_1}) \cdot a \quad \text{over GF(4)}$$

derived from (2.2), where $\langle k \rangle_{10} = \langle k_n \ldots k_1 \rangle_4$ as in Definition 2.12. Using the properties of the Kronecker product, this equation can be solved as follows:

$$a = (S_{k_n} \otimes \ldots \otimes S_{k_1})^{-1} \cdot d \quad \text{over GF(4)} \tag{2.7}$$

$$\begin{aligned} (S_{k_n} \otimes \ldots \otimes S_{k_1})^{-1} &= S_{k_n}^{-1} \otimes \ldots \otimes S_{k_1}^{-1} \\ &= W_{k_n} \otimes \ldots \otimes W_{k_1} \end{aligned}$$

According to Definition 2.12, n-variable q-nary function can have q^n fixed polarity forms. During the synthesis process RM expansions are computed for all polarity forms in order to find the optimal solution. More computationally efficient algorithms than the direct method exist [59, 36, 25]. They exploit certain properties of GF(4) while Green's method is based on a general principle that can be extended to other radices. Denoting the truth vector of a one-variable q-nary function $f(x)$ as $d = \begin{bmatrix} d_0 & \ldots & d_{q-1} \end{bmatrix}^t$, $d_0 \ldots d_{q-1} \in GF(q)$, one can create a matrix equation similarly to the steps (2.4)—(2.6):

$$a = S_k^{-1} d$$

where the polarity number $k = 0 \ldots q-1$. As far as the matrices $S_0 \ldots S_{q-1}$ are invertible in GF(q), corresponding matrices $W_0 \ldots W_{q-1}$ can be found and then used to compute n-variable RM expansions of radix q using (2.7).

For example, in binary case the matrices W_0 and W_1 can be found as follows.

Binary RM expansion of one-variable function takes the form:

$$f(\widetilde{x}) = a_0 + a_1\widetilde{x} \quad \text{over GF(2)}$$

Let the truth vector be defined as $d = \begin{bmatrix} d_0 & d_1 \end{bmatrix}$. For zero polarity case $\widetilde{x} = x$, hence:

$$\begin{aligned} d_0 &= f(0) = a_0 \\ d_1 &= f(1) = a_0 + a_1 \end{aligned}$$

$$a = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{-1} \cdot d = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot d \quad \text{over GF(2)} \tag{2.8}$$

For polarity 1 we have $\widetilde{x} = x + 1$, so the set of equations is changed correspondingly:

$$\begin{aligned} d_0 &= f(0+1) = f(1) = a_0 + a_1 \\ d_1 &= f(1+1) = f(0) = a_0 \end{aligned}$$

$$a = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{-1} \cdot d = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot d \quad \text{over GF(2)} \tag{2.9}$$

From (2.8) and (2.9) we have:

$$W_0 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad W_1 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

The following example illustrates how Green's method can be used to obtain a binary circuit from a set of truth vectors.

Example 2.1. Let binary functions $F_0(y_0, y_1, y_2)$ and $F_1(y_0, y_1, y_2)$ be defined by truth

vectors d_0 and d_1 respectively.

$$\begin{aligned} d_0 &= [11001110]^t \\ d_1 &= [11111101]^t \end{aligned}$$

Binary case matrices W_0 and W_1 shown above can be used to compute the RM expansions. For polarity $k = 0 = \langle 000 \rangle_2$, the coefficient vector for F_0 can be found as follows:

$$a = (W_0 \otimes W_0 \otimes W_0) \cdot d_0 \quad \text{over GF(2)},$$

$$\begin{aligned} a &= \left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \right) \cdot d_0 \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}. \end{aligned}$$

Computing F_1 for the polarity number $k = 1 = \langle 001 \rangle_2$:

$$a = (W_0 \otimes W_0 \otimes W_1) \cdot d_1 = [10000001]^t$$

The functions can be expressed using the computed coefficient vectors as follows.

$$\begin{aligned} F_0(y_0, y_1, y_2) &= 1 + y_1 + y_1 y_2 + y_0 y_1 y_2 \\ F_1(y_0, y_1, y_2) &= 1 + \overline{y_0} y_1 y_2 \end{aligned}$$

The corresponding component-level circuit is shown in Figure 2.11. The circuit has

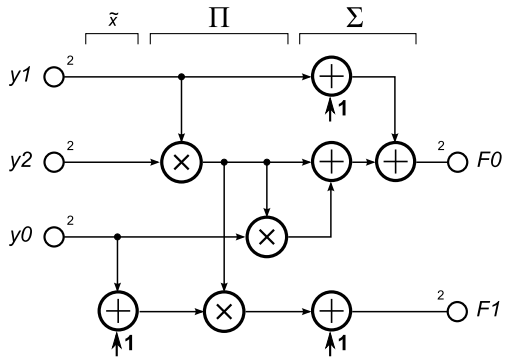


Figure 2.11: Binary RM expansion, component-level schematic.

clearly observed "layers": a layer of sums (marked as Σ), a layer of products (Π) and a layer of polarities ($\widetilde{x}$). The number of components has been also optimised by reusing the results of some operations. For example, the term y_1y_2 is reused three times.

Example 2.2. Let two-variable quaternary function be defined by the truth vector $d = [BBAABB1ABBAABB1A]^t$.

Quaternary RM expansion gives

$$a = (W_0 \otimes W_2) \cdot d \quad \text{over GF(4)},$$

$$
\begin{aligned}
a &= \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & B & A \\ 0 & 1 & A & B \\ 1 & 1 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 1 & 0 \\ B & A & 0 & 1 \\ A & B & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \right) \cdot d \\
&= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B & A & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & B & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & B & 0 & 0 & 0 & A & 0 \\ 0 & 0 & 0 & 0 & B & A & 0 & 1 & A & 1 & 0 & B & 1 & B & 0 & A \\ 0 & 0 & 0 & 0 & A & B & 0 & 1 & 1 & A & 0 & B & B & 1 & 0 & A \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & B & B & B & B & A & A & A & A \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & A & 0 & 0 & 0 & B & 0 \\ 0 & 0 & 0 & 0 & B & A & 0 & 1 & 1 & B & 0 & A & A & 1 & 0 & B \\ 0 & 0 & 0 & 0 & A & B & 0 & 1 & B & 1 & 0 & A & 1 & A & 0 & B \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & A & A & A & A & B & B & B & B \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ B & A & 0 & 1 & B & A & 0 & 1 & B & A & 0 & 1 & B & A & 0 & 1 \\ A & B & 0 & 1 & A & B & 0 & 1 & A & B & 0 & 1 & A & B & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} B \\ B \\ A \\ A \\ B \\ B \\ 1 \\ A \\ B \\ B \\ A \\ A \\ B \\ 0 \\ B \\ A \end{bmatrix} = \begin{bmatrix} A \\ 1 \\ 1 \\ 0 \\ A \\ 0 \\ 0 \\ A \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix},
\end{aligned}
$$

for polarity $k = 2 = \langle 0A \rangle_4$, and the function can be expressed as follows:

$$F(x_0, x_1) = A + \ddot{x_0} + \ddot{x_0}^2 + Ax_1 + A\ddot{x_0}^3 x_1 + x_1^2 + \ddot{x_0}^3 x_1^2,$$

where $\ddot{x_0} = x_0 + A$.

The circuit is shown in Figure 2.12. The layered structure of a quaternary circuit includes a layer of coefficients (a) and a layer of powers (x^n).

2.3 Summary

The chapter has given the brief explanation of the hardware attacks and countermeasures. Power balancing has been selected among the listed countermeasures to the

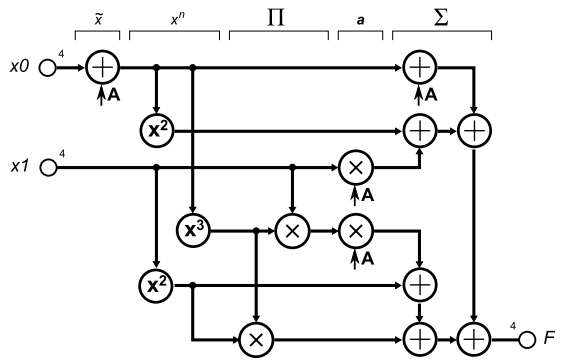


Figure 2.12: Component-level schematic for uniform radix (quaternary) RM expansion.

side-channel hardware attacks as the most universal one. The basic principle of power balancing is to make a circuit that consumes the same amount of power regardless of what data is being processed. One of the possible ways to make the circuit power balanced is to encode it using m-of-n codes. The logic components working with encoded signals also must preserve the property of power balancing – they must have data-independent switching activity and power. However, it is also reasonable to allow slightly compromised (relaxed) balancing in order to reduce component costs.

The second part has presented the theory of Galois fields, described the properties of the fields and operations over them. Reed-Muller expansions have been introduced, and a direct method of computation has been defined. The theory is a foundation for the research presented in the following chapters.

Chapter 3

Baseline Research

The purpose of this chapter is to provide an application-specific foundation for the research. More specificly, it addresses to the following tasks:

- to give an overview of the possible MVL design solutions w.r.t. the security application;
- to present an alternative approach, so the main theory can be tested against it;
- to define the range of benchmarks.

3.1 Secure Design Flow

Considering the trends in countermeasure methodology described in Chapter 2, one can conclude that security requirements affect the design flow as follows.

- Data signal representations other than traditional binary (single-rail) signals are required, this may include multi-valued signals. This requirement implies advanced logic synthesis and affects the data path.
- Control path has to be synthesised using asynchronous system design techniques. For asynchronous designs it is possible to develop the data path and the control path concurrently since in general the control path does not rely on the timing properties of the combinational logic.

· Physical layout may also have an impact on the security properties, since (i) it affects wire capacitance and consequently power balancing and (ii) floor-planning defines electromagnetic emission patterns.

Independent consideration of the latter approaches allows more efficient combinations of countermeasures to be used. The whole design environment can be split into three parts: data path synthesis, control path synthesis and layout. Traditionally design environments consist of multiple tools, each is used for a different task. Thus partitioning the flow should not be a problem. However certain compatibility issues take place, so it is often not possible to create direct connection between tools from different design environments.

The conversion driven design (CDD) approach is also considered. It suggests conversion of existing insecure circuits applying security features on top of the previously designed netlists. In this case the design environment structure uses conversion tools instead of synthesis tools. This may include tools for desynchronising clocked circuits and tools for converting signal representation in existing binary (single-rail) data paths.

This section presents the discussion on the possibilities of security applied design with respect to both synthesis and conversion. The data path and control path tools are discussed independently and analysed from the perspective of interfacing between them in order to create a solid secure design flow.

3.1.1 Data path synthesis

The only signal representation supported by the industrial logic synthesis tools is the traditional binary encoding, which is also called single-rail since one wire represents one bit of data. Consequently, this lack of support often discourages designers from using power balanced protocols. In order to make the proposed design flow more attractive to engineers we suggest using RTL-implemented power-balanced logic components.

As it was mentioned bfore, there are two possible ways of obtaining the security-

aware encoded data path: the secure data path can be obtained by transforming the existing insecure data path (conversion) or it can be synthesised from the specification with respect to the application requirements (synthesis).

Conversion approach is believed to be more convenient for industrial development as it requires minimal modification to the design environment. The simplest way to apply power balancing is to map previously synthesised binary logic directly into power-balanced dual-rail components. This can be done using Verimap tool [67].

A significant drawback of dual-rail versus other m-of-n codes is the increased power consumption. An efficient solution to this problem is to use multi-valued logic (MVL) instead of binary. Due to the properties of m-of-n codes, and especially 1-of-n codes, higher radix signals produce less switching activity of wires reducing the power consumption. An attempt to use the conversion approach for MVL synthesis has been made in Section 3.2.

In terms of the design flow, as can be observed from Figure 3.1, the approach has a number of drawbacks. First, a design compiler has to be used twice: to synthesise the original circuit and to merge the converted data path with the control path. This complication may negatively impact on the testability of the design, since it is very difficult to reflect possible errors back to the original high-level description. Second, since conversion at this stage is applied only to the data path, there should be some method to extract it from the flat gate-level netlist. Normally this is done by a conversion tool, but the very idea is rather weak.

All these bottlenecks can be avoided if MVL is applied not at the post-synthesis stage, but at the pre-synthesis stage. Certain modules in the initial HDL description can be replaced with the modules preprocessed (synthesised) using special tools. The paradigm of accommodating different radices within a circuit has been evolved in the work presented in Chapter 4, which uses Reed-Muller expansions based on Galois field arithmetic. Since it considers the notion of mixed radices from the highest level of mathematical representation, the synthesised logic does not require internal signal conversion and shows significant improvement in power and area. A detailed

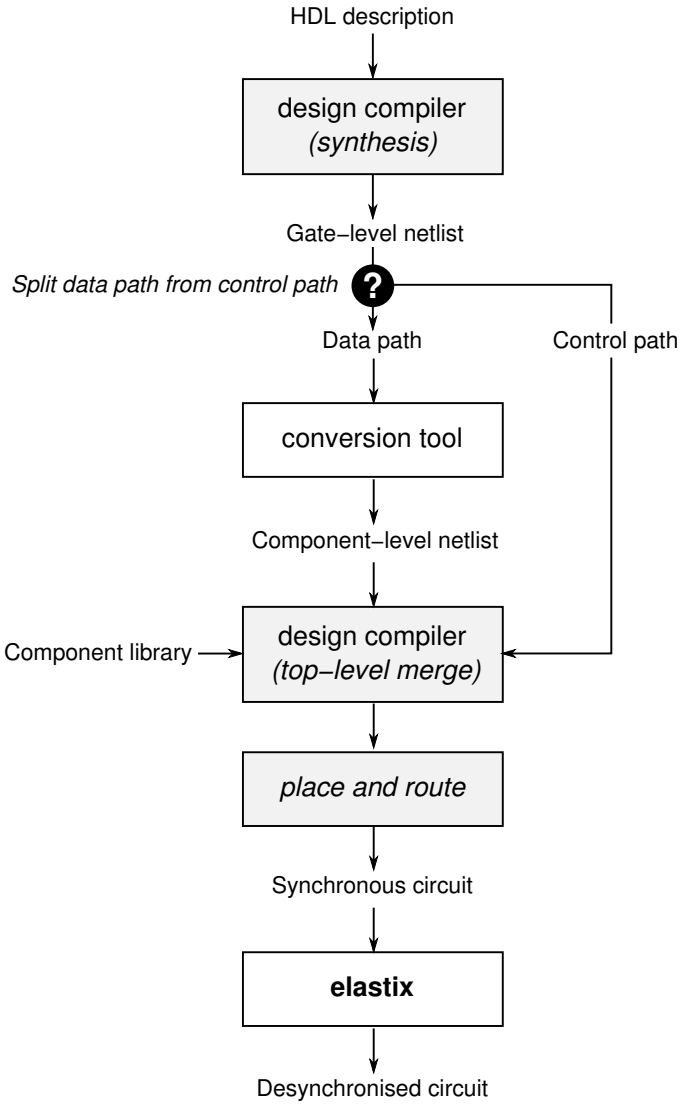


Figure 3.1: CDD-based design flow

description of the tool and its features is given in Chapter 5. In terms of the design flow, the tool allows the designer to choose which modules to synthesise using Reed-Muller approach, i.e. to use power balancing locally if needed. It can be emerged into the flow as shown in Figure 3.2, the rest of the tools used in the illustrated flow are described further in Subsection 3.1.2.

Both flows are application driven. They have an explicit component level netlist at some point of the design process. Any circuit optimisation must be applied at that stage; the following run of the design compiler is supposed to be an exact mapping.

3.1.2 Control path synthesis

The flexibility of the secure design flow implies an "unconstrained" choice of the security countermeasures, so they can be used in different combinations if needed. It is important to take into consideration the case of power balanced but synchronous designs that may be sufficient for achieving certain level of security. Most of the custom logic synthesis tools can output Verilog format, thus customised combinational logic can be merged with the control path using standard EDA tools. However, as the proposed design flow implies security at all levels of the design, asynchronous methods should be used instead.

Desynchronised circuits A conversion method for the control path has been described in [19]. The principal idea is to replace clocked registers in synchronous designs with handshake registers. Elastix tool [72] implements this approach. The input of the tool is a finalised synchronous design. Hence the step of merging customised combinational logic with a clocked control path is to be done before the stage of desynchronisation. The possibility to achieve the required secure properties without redesigning the whole system "from scratch" motivates designers to use the desynchronisation approach. However, with respect to the conversion flow shown in Figure 3.1, the drawbacks of the data path conversion approach may impair the benefits of the Elastix flow.

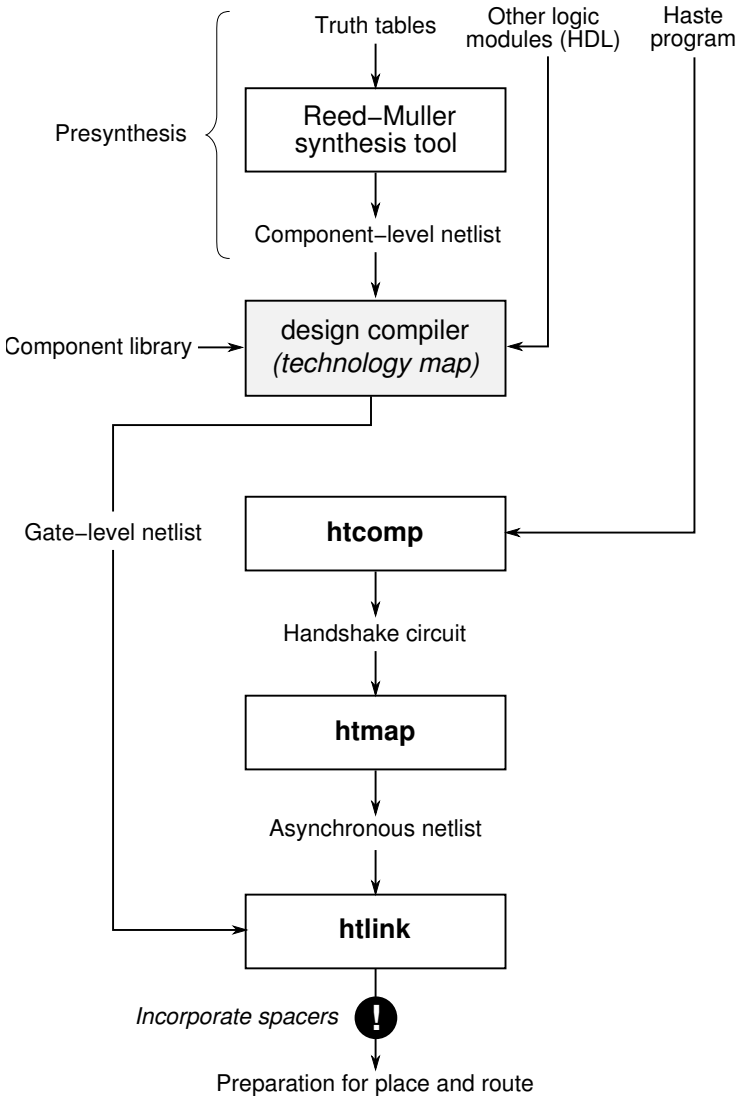


Figure 3.2: Reed-Muller + Haste synthesis flow

Handshake-based asynchronous circuits Asynchronous circuits use the request-acknowledge interaction between components to control the system. This way of signalling is called *handshake protocol*, and it can be implemented in different ways. The tools for asynchronous control path synthesis use netlists of handshake components as an intermediate state of the design.

The most widely known toolkits for the asynchronous development are Balsa toolkit [22] and TiDE [2].

Balsa-based environment uses *balsa-c* compiler, which compiles a Balsa description into the netlist of handshake components, and *balsa-netlist* gate-level mapper. The major drawback of this environment with respect to the secure design flow is that it uses a predefined set of components, so it is impossible to use custom data representations in its standard workflow. The only solution is to build the control path in a separate module and then merge modules using third-party software [49, 50].

The TiDE design environment, developed by Handshake Solutions, is based on the Haste description language and uses *htcomp* compiler to synthesise handshake component netlists; *htmap* substitutes abstract handshake components with gate-level modules; *htlink* is used to merge the control path with presynthesised (custom) combinational logic provided in a form of gate-level netlist.

At the moment TiDE appears to be the most flexible and industry-aware asynchronous design environment, and it has already been used in security applications. Figure 3.2 illustrates how the data path synthesis can be interfaced into TiDE.

Unfortunately, in terms of power balancing there is a conflict that has to be resolved. It is known that an asynchronous system can be designed using a single-rail bundled data protocol or m-of-n encoded spacer protocol [68]. In terms of security, the spacer protocol is essential for power balancing as it guarantees equal number of wires switching per period. Indeed, since an m-of-n encoded signal implies m wires to be set to 1, exactly m wires switch from 0 to 1 when the data arrives and reset back to 0 on spacer. Equalised switching of wires is fundamental for power balanced circuits, and exclusion of the spacer from the protocol also removes this property.

Based on the single-rail data format, TiDE uses bundled data protocol and cannot be directly used for m-of-n encoded power balanced designs. However, in systems which employ bundled data protocol the spacer may be emulated (enforced) by explicitly alternating data and spacers at the inputs. Simple for pipelined systems, this approach becomes rather difficult for the systems with loops in the structure. However the known solution is to use master-slave registers [67, 69], so one stores spacer, while another stores data and vice versa. This kind of behaviour can be described in the high-level code, but this requires in-depth knowledge of asynchronous design technology.

In contrast, the presented research attempts to minimise the designer's effort and automate the process. Our proposal is to adjust the final bundled data asynchronous circuits by directly replacing memory cells with custom registers that inject spacers into existing paths. For example, the principle of "wagging" registers [73] implements spacer and data alternation using two flip-flops connected in parallel via multiplexer (in contrast to sequentially connected master-slave registers).

3.1.3 Physical design

Layout issue is highly important for power balancing since it defines the capacitance of wires and global electromagnetic emission patterns. In m-of-n codes, considering that each data signal is transferred using a set of wires, balanced switching activity makes sense only if the switching energies of wires in each bus are equal. Consequently the secure design flow should guarantee equalised wire lengths and fanout, i.e. the parameters affecting the switching energy of gates.

In general this can be done using advanced scripting in standard industrial layout tools. However the best way to implement power balanced layout is to route buses as parallel sets of wires, which is not supported by common tools. Pulsic software [3] specialising in the layout for custom designs can perform such routing of wires, so it is advised for security applications.

3.2 Conversion Driven Mixed Radix Design

This section presents the design approach based on reusing the existing insecure designs in order to produce secure higher radix circuits. The goal of this approach is to achieve a tight integration with the conventional EDA flows with low algorithmic complexity. Our justification and reasoning stems from the following facts:

- Moving away from the RTL design flow is frequently frowned upon by industry;
- Existing EDA tools are mature, known and time-proven;
- MVL synthesis methods employ computationally expensive algorithms instead of reusing the computational power of existing tools.

3.2.1 Conversion basics

The problem addressed in this section can be characterised as follows. The original binary datapath is given as a structural HDL netlist, where datapath is defined as logic gates without registers or combinational loops. The goal of the conversion is to produce an equivalent higher radix circuit, e.g. radix $q > 2$. Since at this stage the actual encoding or technology is not considered, q-nary circuit is represented by a netlist of q-nary components.

The set of available components can be defined in many ways, however the convenient way for the conversion approach is to obtain q-nary components by grouping binary gates together. Formally, for radix $q = 2^n$ and given binary-to-q-nary radix mapping $r : \{0,1\}^n \rightarrow \{0,\ldots,2^n-1\}$, q-nary function f_q can represent a group of n binary functions $f_{2,1},\ldots,f_{2,n}$ as

$$f_q(y) = f_q(r(x_1,\ldots,x_n)) = r(f_{2,1}(x_1,\ldots,x_n),\ldots,f_{2,n}(x_1,\ldots,x_n))$$

for any $y \in \{0,\ldots,2^n-1\}$, $x_1,\ldots,x_n \in \{0,1\}$ and $y = r(x_1,\ldots,x_n)$.

Similar problem adressed to programmable logic arrays (PLA's) has been solved

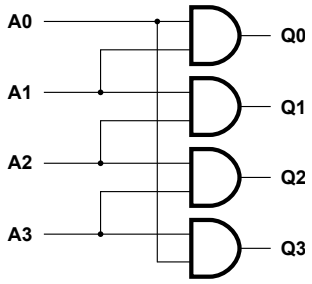


Figure 3.3: Example of a binary circuit that cannot be efficiently converted into a higher radix circuit.

in [64]. For the case of the gate-level netlist, we looked into the possibilities of simple structural transformations of the circuit. Subsection 3.2.2 gives more details on the subject.

The major challenge of the conversion approach is the fact that grouping of gates (and corresponding signals) is not always possible. The example in Figure 3.3 illustrates the case. Assuming that the target radix is 4, grouping of gates Q0 and Q1 implies grouping of inputs {A0, A1}, {A1, A2}, and grouping of the remaining gates Q2 and Q3 requires {A2, A3} and {A3, A0} to be grouped too. But the signals A0 and A2 are already used. The same happens if one attempts grouping any other combination of gates in this example.

The original structure usually causes reshuffling and splitting of higher radix data, hence grouping of all gates in the circuit cannot be globally efficient. The circuit becomes partially binary and partially multi-valued, i.e. mixed radix. The result of conversion is shown in Figure 3.4 and consists of binary and quaternary blocks connected through a set of signal converters. We distinguish two types of signal converters: *splitters*, which split higher radix signals into lower radix signals, and *mixers*, which mix lower radix signals into a higher radix signals. After the circuit is technology mapped and encoded, signal converters ensure the link between different data representations.

Conversion is also possible for radices other than 2^n; in this case certain overhead is introduces by radix mapping with its non-matching cardinality of domain and

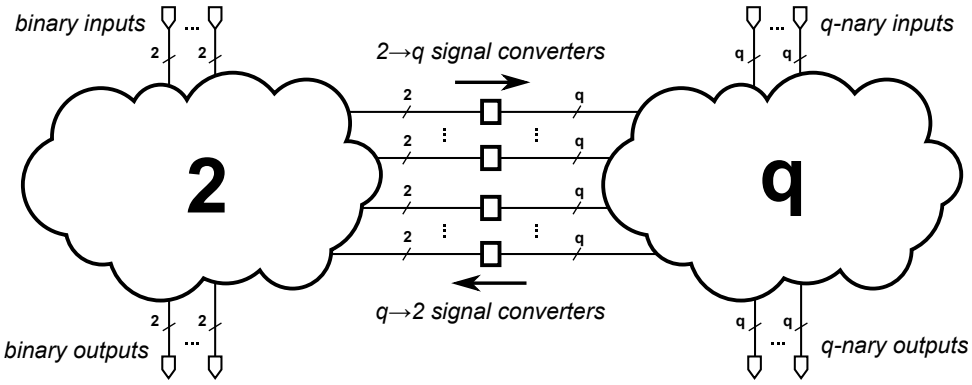


Figure 3.4: Mixed radix circuit as a result of binary-to-q-nary conversion.

co-domain. This chapter explores the simplest case of radix conversion: binary to quaternary conversion, which implies grouping of binary gates by two.

A generic outline for the proposed conversion technology can be described as follows. The algorithm starts by "transferring" gates from the binary part of the circuit to the quaternary part, initially empty, by grouping them into pairs. During this phase the conversion uses technology independent (abstract) binary and quaternary components, thus adding a component level of abstraction to the design flow. After all possible grouping is done the circuit can be mapped into a gate-level netlist replacing components with real cells using specific encoding and library.

The way the gates are grouped determines the efficiency of the conversion, therefore the conversion problem corresponds directly to the gate grouping problem described in the following subsections.

3.2.2 Types of gate grouping

For 2n-bit binary circuits there is an intuition to group higher and lower bits of each signal pair, as shown in Figure 3.5(a), to form a n-signal quaternary circuit. Certain gates which violate bitwise regularity of the original circuit will remain ungrouped forming a binary part of the resultant mixed radix circuit.

Although the intuition behind the bitwise approach is straightforward, automatically distinguishing even and odd bit parts of the given netlist is computationally

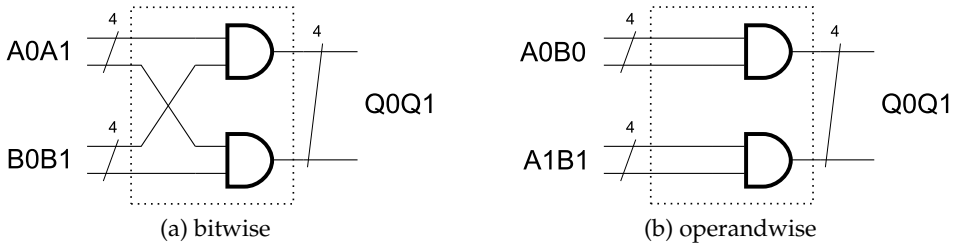


Figure 3.5: Types of gate groupings (considering the same original circuit).

complex or, in certain cases, infeasible. For example, S-box circuits [9, 8] tend to reshuffle input data, thus input signals have no bitwise meaning. Natural for the synthesis from functional specification, bitwise grouping is not suitable for structural netlist transformations, especially when the original structure requires data "shifting" between bitwise parts of the circuit. This necessitates finding a more efficient way of gate grouping with respect to the CDD approach.

Asuming that the original circuit consists of two-input standard cells (AND, OR, XOR gates) and inverters, it is possible to group inputs of any gate into one quaternary signal. Outputs of a given pair of gates can also be grouped into a quaternary signal. This produces an *operandwise grouping* of gates illustrated in Figure 3.5(b).

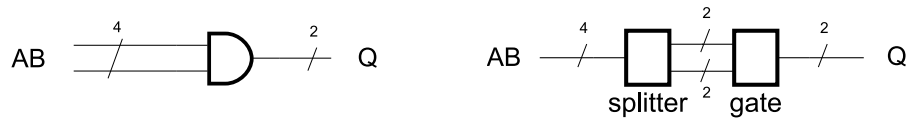
Due to the nature of operandwise grouping, any quaternary signal x can be rewritten as a pair of its original signals, $x = \langle s_1, s_0 \rangle_4$. For two quaternary signals $x = \langle s_1, s_0 \rangle_4$ and $y = \langle t_1, t_0 \rangle_4$, two binary functions A and B can form a quaternary operandwise operation $\langle A, B \rangle_4$ shown in (3.1) and clarified in Table 3.1.

$$Q_{AB}(x, y) = \langle A(s_1, s_0), B(t_1, t_0) \rangle_4 \tag{3.1}$$

In a case when the output of a quaternary gate has to be split again into binary signals an insertion of a splitter can be avoided using incomplete operandwise grouping, i.e. a grouping when gate inputs are grouped, but the output remains binary as shown in Figure 3.6(a). A *Q/B gate* or $4 \rightarrow 2$ *gate* is a gate with one quaternary input and one binary output. Due to their semantical meaning clarified in Figure 3.6(b), Q/B gates can eliminate unnecessary splitters during the conversion, as shown in Figure 3.7.

Table 3.1: Bitwise and operandwise quaternary operations example

x	y	bitwise AND	operandwise AND-AND
0	0	$0 \wedge 0 = 0$	$\langle 0 \wedge 0, 0 \wedge 0 \rangle_4 = \langle 0, 0 \rangle_4 = 0$
0	1	$0 \wedge 1 = 0$	$\langle 0 \wedge 0, 0 \wedge 1 \rangle_4 = \langle 0, 0 \rangle_4 = 0$
0	2	$0 \wedge 2 = 0$	$\langle 0 \wedge 0, 1 \wedge 0 \rangle_4 = \langle 0, 0 \rangle_4 = 0$
0	3	$0 \wedge 3 = 0$	$\langle 0 \wedge 0, 1 \wedge 1 \rangle_4 = \langle 0, 1 \rangle_4 = 1$
1	0	$1 \wedge 0 = 0$	$\langle 0 \wedge 1, 0 \wedge 0 \rangle_4 = \langle 0, 0 \rangle_4 = 0$
1	1	$1 \wedge 1 = 1$	$\langle 0 \wedge 1, 0 \wedge 1 \rangle_4 = \langle 0, 0 \rangle_4 = 0$
1	2	$1 \wedge 2 = 0$	$\langle 0 \wedge 1, 1 \wedge 0 \rangle_4 = \langle 0, 0 \rangle_4 = 0$
1	3	$1 \wedge 3 = 1$	$\langle 0 \wedge 1, 1 \wedge 1 \rangle_4 = \langle 0, 1 \rangle_4 = 1$
...	...	...	...
3	0	$3 \wedge 0 = 0$	$\langle 1 \wedge 1, 0 \wedge 0 \rangle_4 = \langle 1, 0 \rangle_4 = 2$
3	1	$3 \wedge 1 = 1$	$\langle 1 \wedge 1, 0 \wedge 1 \rangle_4 = \langle 1, 0 \rangle_4 = 2$
3	2	$3 \wedge 2 = 2$	$\langle 1 \wedge 1, 1 \wedge 0 \rangle_4 = \langle 1, 0 \rangle_4 = 2$
3	3	$3 \wedge 3 = 3$	$\langle 1 \wedge 1, 1 \wedge 1 \rangle_4 = \langle 1, 1 \rangle_4 = 3$



(a) Q/B gate as an incomplete operandwise group

(b) structural decomposition

Figure 3.6: Understanding Q/B gates.

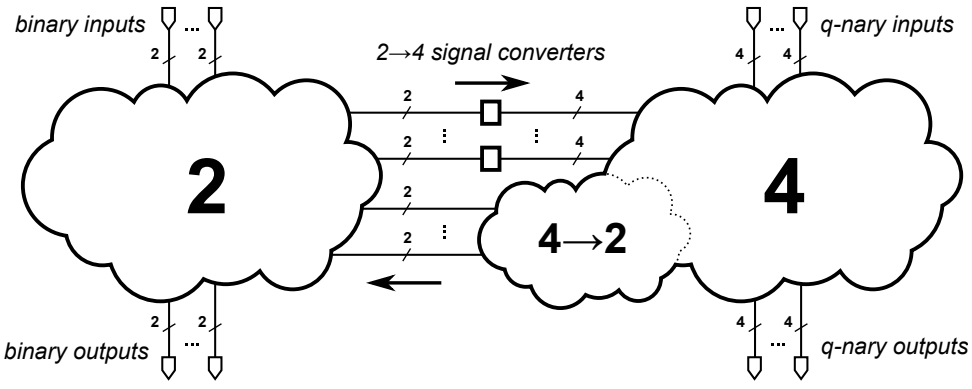


Figure 3.7: Mixed radix circuit using Q/B gates

3.2.3 Grouping based on bitwise regularity

Consider a binary circuit, which perform calculations on 2-bit values. Assuming that separate calculations of both bits are relatively similar, one can determine for any gate in the lower bit part its equivalent in the higher bit part. A distinguished pair of such relative gates can form a quaternary gate. For n-bit binary circuits, if $n > 2$, gates can be grouped in similar way but merging even and odd bit parts of the circuit. Parts of the circuit that cannot be distributed between certain bit parts remain binary.

Although the intuition behind the bitwise approach is straight-forward, automatically distinguishing even and odd bit parts of the given netlist is computationally complex or, in certain cases, infeasible. For example, S-box circuits [9, 8] tend to reshuffle input data, thus input signals have no bitwise meaning. However, circuits displaying bitwise regularity can be converted using this approach if the information on the bitwise meaning of the input and output signals of the datapath is supplied, i.e. input and output ports are initially grouped into pairs. It is possible to automate port grouping using a naming convention.

The algorithm shown in Algorithm 1 implies bitwise gate grouping using given netlist and port grouping information. Here G is defined as a set of binary gates; initially it contains all gates of the original circuit. Each gate $g \in G$ has a preset $I(g) = \{i_0(g), \ldots, i_{n-1}(g)\}$, i.e. other gates or circuit ports connected to the inputs of g; n is the number of inputs of g. P is a set of grouped gates or circuit ports (pairs). Since port grouping specification is given, P initially contains paired circuit ports. Please note that the order of items in a bitwise pair is important.

Definition 3.1. *Bitwise regularity ratio* (BRR) υ for the given group of gates is a characteristic showing how many quaternary links the group can form w.r.t. current state of P. In other words, for a bitwise group $p = \{g_1, g_2\}$ BRR can be calculated as follows:

Algorithm 1 Grouping based on bitwise regularity

```
all ports are assumed to be already grouped
repeat:
    υmax = 0
    N = size of G
    for i = 0 to N−2:
        for j = i+1 to N−1:
            if g_i ∈ I(g_j) or g_j ∈ I(g_i): skip this pair
            υ = regularity ratio for group {g_i, g_j}
            if υ > υmax:
              υmax = υ
              pmax = {g_i, g_j}
            end if
        end for
    end for
    if υmax > 0:
        add pmax to P
        remove gates in pmax from G
    end if
until there are no more pairs with υ > 0
```

$$\upsilon(p) = \frac{\upsilon_{out} + \sum_{j=1}^{n} \upsilon_j}{1+n}$$

where n is a number of gate inputs, $n = n(g_1) = n(g_2)$, and

$$\upsilon_{out} = \begin{cases} 1, & \text{if there exist such } k \text{ and } \{e_1, e_2\} \in P \text{ that } g_1 = i_k(e_1), g_2 = i_k(e_2) \\ 0, & \text{otherwise} \end{cases}$$

$$\upsilon_j = \begin{cases} 1, & \text{if there exist such } \{e_1, e_2\} \in P \text{ that } e_1 = i_j(g_1), e_2 = i_j(g_2) \\ 0, & \text{otherwise} \end{cases}$$

In these equations we assume that gates g_1 and g_2 are similar, i.e. they represent the same function and have equal number of inputs n. Considering similarity of bitwise circuit parts the gates corresponding to the same operation are also supposed to be similar. However, this constraint is not essential if we have a methodology of grouping

non-similar gates.

BRR is used as an estimation criterion in breadth-first search. As each iteration of outer loop in Algorithm 1 adds new pair in P, BRRs of gate pairs change over time. If the search reveals several maximum regular pairs, the algorithm uses the first encountered one instead of searching through all possible varieties; this feature is treated as disadvantage which in certain cases can lead to inefficient results.

The described algorithm has polynomial complexity $O\left(N^3\right)$ not considering calculations of BRR, where N is the number of binary gates in the initial circuit. BRR calculation for one pair has a complexity $O\left(\mathfrak{n}M + \mathfrak{n}M\right)$, where $\mathfrak{n}$ is an average number of gate inputs and M is a size of P at the moment. For $\mathfrak{n} = 2$ and linearly growing P we have total algorithmic complexity $O\left(N^4\right)$. In terms of CDD this computational cost is rather expensive; VLSI design presumably requires conversion of datapath blocks with $N > 500$.

Example 3.1. Consider a 2-bit full adder shown in Figure 3.8. Initially P consists of pairs {A0, A1}, {B0, B1}, {Q0, Q1} due to the bitwise meaning of port signals. Ports C and CC have no pairs and remain dual-rail. Conversion using Algorithm 1 can be done in the following steps:

Iteration 0 maximum regular pair is {g00, g01} – all inputs can form quaternary links with input ports.

Iteration 1 maximum regular pair is {g10, g11} – the same reason.

Iteration 2 {g20, g21} – it can form a quaternary connections with {g00, g01} and with output port {Q0, Q1}.

Iteration 3 {g30, g31} – it can form a quaternary connection with {g00, g01}.

Iteration 4 {g40, g41} – it can form a quaternary connection with {g10, g11}.

There are no unpaired gates remaining. Please note that each iteration here is the iteration of outer loop, which in its turn searches through $4N\log_2^2 N$ iterations. Resultant circuit after insertion of signal converters is shown in Figure 3.9.

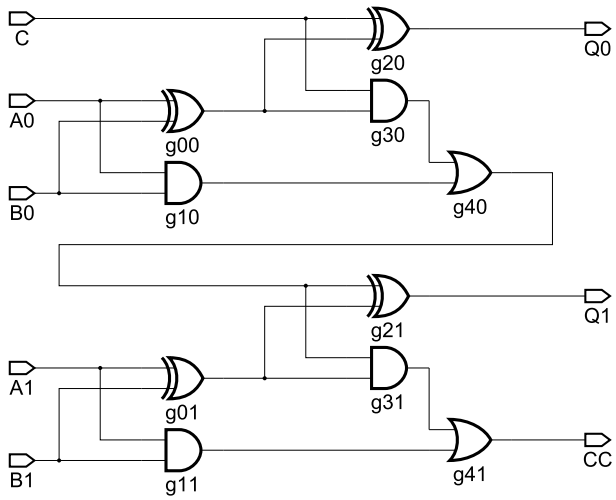


Figure 3.8: Example original single-rail circuit: 2-bit adder.

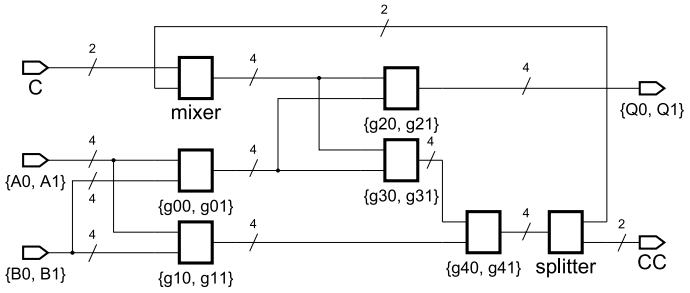


Figure 3.9: 2-bit adder converted using bitwise regularity approach; gates are shown as “black boxes”.

The described example demonstrates another drawback of the bitwise approach: it can form combinational loops in the resultant circuit. Without special consideration and additional completion detection RTZ-aware components [80] cause a spacer deadlock in these loops. Consider the mixer in Figure 3.9. Assume that the whole circuit is initially reset to spacer value. Incoming data from the port C cannot pass through the mixer because a spacer from the looped wire (an internal carry) blocks it. On the other hand, if the mixer uses a protocol allowing the data to propagate regardless to spacers, the circuit will produce invalid data output because a valid result is available only after the second iteration.

In spite of a number disadvantages revealed in the approach, the grouping based on bitwise regularity shows several positive features:

- Port bits are not reshuffled with respect to bitwise meaning of input and output signals, which is useful from the large scale design view.
- Resultant circuits have a structure similar to the original, which allows using placement suggestions from the single-rail equivalent.

3.2.4 Grouping based on binary trees

The operandwise grouping suggests a binary trees approach considering gates of the circuit to be tree-nodes and their inputs to be child branches. As it was mentioned before, the given datapath circuit contains no loops. However, a pure tree-like structure can be blocked by gates with multiple fanout. The tree size can be reduced by recursive operandwise grouping of child nodes for each gate in binary trees within the circuit. This grouping causes all signals in tree-like structures to become 1-of-4 encoded, but "blocked" parts of the circuit remain binary and go to dual-rail.

Formally, a circuit is considered to be a set of entities E; each entity has a type of *input*, *output*, or *gate*. Input and output entities are circuit ports. Each non-input entity $e \in E$ has a preset $I(e) = \{i_0(e), \ldots, i_{n-1}(e)\}$, i.e. entities connected to the inputs of e. Due to declared constraints $n = 2$ for gates, and $n = 1$ for outputs. Set P is a set of

gate groups (pairs). It represents 1-of-4 encoded part of the circuit. In addition there is a parameter θ_e that stands for encoding of entity $e \in E$ and can be either dual-rail or 1-of-4. It is used for automated insertion of signal converters (splitters and mixers) into the final circuit.

The proposed algorithm contains three phases as shown in Algorithm 2. To avoid recursion the grouping is done in two search passes through E (phase 1 and 2). The first phase ignores gate fanouts and groups all signals considering the whole circuit as a binary tree. This leads to duplication of certain gates. The second phase analyses the duplicates and discards the groups which lead to duplication. The last phase reconstructs correctness of links between gates by inserting signal converters where appropriate. Splitters are reduced to Q/B gates.

There is no simple solution to estimate optimal grouping of outputs but to search through all possibilities. However, due to the nature of binary trees approach the grouping of outputs has minor influence to the structure comparing with the whole circuit. Therefore a suggestion to group any pair of outputs is accepted.

The computational complexity of the algorithm is $O\left(N^2\right)$, where N is the size of E. The algorithm is highly modular; one can add more passes to the algorithm to increase efficiency of the conversion. However it can produce significant "fractioning" of dual-rail and 1-of-4 parts of the circuit increasing the number of signal converters required.

Example 3.2. Consider a 2-bit adder shown in Figure 3.8. Preset gates of the gate g40 can form the group {g30, g10}; similarly ports A0 and B0 are grouped as inputs to g00 or g10. Considering all gates we can make the following grouping: {A0, B0}, {A1, B1}, {g30, g10}, {g31, g11}, {g40, g01}. Preset of gates g20 and g30 cannot be grouped because of the different entity types (input C cannot be grouped with the gate g00). The second phase should cancel signal duplicating gate groups, but in this case nothing is to be cancelled. Indeed, in spite of the fact that some gates and ports have fanout > 1, they are not shared between different groups and do not lead to signal duplication. Finally, randomly selected output grouping {Q0, Q1} leads to the grouping {g20, g21}. Resultant

Algorithm 2 Conversion based on binary trees

Phase 1: group all gate inputs.

```
for each gate g in E:
    if i0(g) and i1(g) are of the same type:
        group {i0(g), i1(g)} and add to P
end for
```

Phase 2: discard groups containing gates with fanout > 1.

```
for each non-output entity e in E:
    u = how many groups share e
    if u > 1: remove groups containing e
    else if u = 1: set θe to 1-of-4
    else set θe to dual-rail
    end if
    if e is gate and group {i0(e), i1(e)} ∉ P:
        remove all groups containing i0(e) or i1(e)
end for
```

Phase 3: insert mixers and Q/B gates.

```
for each gate g in E:
    if θi(g) is 1-of-4 and θg is dual-rail:
        set type of g to Q/B gate
    else if θi(g) is dual-rail and θg is 1-of-4:
        insert mixer before g
end for
```

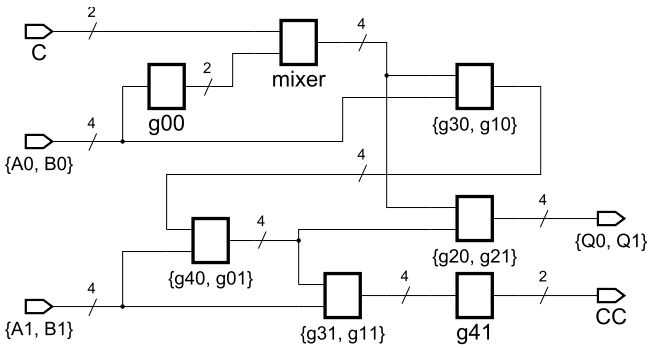


Figure 3.10: 2-bit adder converted using binary trees approach; gates are shown as "black boxes".

circuit is shown in Figure 3.10. Gates g00 and g41 are Q/B gates.

3.2.5 Component implementation

Moving from abstract binary and quaternary signals to specific encodings, dual-rail and 1-of-4 correspondingly, CDD requires proper definition of mixed radix components.

The resultant circuit should be consistent with RTZ protocol, i.e. its components should satisfy the *spacer condition*, which means that the output of a component must go to a spacer value (NULL) if any of arguments have spacer value.

In this section we use the following definition of polarity.

Definition 3.2. For a binary function $F(x)$, its *polarity representation* is $F^{\langle k \rangle}(x)$, such that $F^{\langle k \rangle}(x) = F(x)$ for $k = 0$, and $F^{\langle k \rangle}(x) = \overline{F}(x)$ for $k = 1$.

From (3.1) for each wire q_i of 1-of-4 encoded $Q_{AB} = \{q_3, \ldots, q_0\}$ w.r.t. spacer condition we have:

$$q_i = \begin{cases} 0, & \text{if } x = \text{NULL or } y = \text{NULL} \\ A^{\langle i_1 \rangle}(x) \cdot B^{\langle i_0 \rangle}(y), & \text{otherwise} \end{cases} \tag{3.2}$$

Here i_0 and i_1 are the bits of binary representation of integer i, in other words $i_{10} = \langle i_1, i_0 \rangle_2$.

Let

$$f_F^{\langle k \rangle}(x) = \begin{cases} 0, & \text{if } x = \text{NULL} \\ F^{\langle k \rangle}(x), & \text{otherwise} \end{cases} \tag{3.3}$$

then (3.2) becomes:

$$q_i = f_A^{\langle i_1 \rangle}(x) \cdot f_B^{\langle i_0 \rangle}(y) \tag{3.4}$$

According to Table 2.1, binary signals s_1, s_0 can be expressed using certain wires $x_3, \ldots, x_0$ of 1-of-4 encoded quaternary $x = \langle s_0, s_1 \rangle_4$ as follows:

$$\begin{aligned} s_0 &= x_3 + x_1 & \overline{s}_0 &= x_2 + x_0 \\ s_1 &= x_3 + x_2 & \overline{s}_1 &= x_1 + x_0 \end{aligned} \tag{3.5}$$

Applying (3.5) and the spacer condition (w.r.t. 1-of-4 code) to (3.3) we have:

$$f_F^{\langle k \rangle}(x_3, \ldots, x_0) = F^{\langle k \rangle}(x_3, \ldots, x_0)(x_3 + \ldots + x_0)$$

Total set of functions $f_F^{<k>}(x)$ for the standard functions $F = \{\text{AND}, \text{OR}, \text{XOR}\}$:

$$\begin{aligned} f_{\text{AND}}^{\langle 0 \rangle}(x) &= x_0 + x_1 + x_2 & f_{\text{AND}}^{\langle 1 \rangle}(x) &= x_3 \\ f_{\text{OR}}^{\langle 0 \rangle}(x) &= x_0 & f_{\text{OR}}^{\langle 1 \rangle}(x) &= x_1 + x_2 + x_3 \\ f_{\text{XOR}}^{\langle 0 \rangle}(x) &= x_0 + x_3 & f_{\text{XOR}}^{\langle 1 \rangle}(x) &= x_1 + x_2 \end{aligned} \tag{3.6}$$

For security application the circuit components should have balanced switching to guarantee data independent power consumption. This can be done using identical gates with certain inputs connected to the ground. Thus $f_F^{\langle k \rangle}(x)$ should be symmetric with regard to k, and representations $f_{\text{OR}}^{\langle 0 \rangle}(x) = x_0 + 0 + 0$ and $f_{\text{AND}}^{\langle 1 \rangle}(x) = x_3 + 0 + 0$ should be used instead.

Example 3.1. From (3.4) and (3.6), considering balanced switching, one can derive a set of equations (3.7) defining an operandwise 1-of-4 AND-XOR group.

$$\begin{aligned} q_0 &= (x_0 + x_1 + x_2)(y_0 + y_3) \\ q_1 &= (x_0 + x_1 + x_2)(y_1 + y_2) \\ q_2 &= (x_3 + 0 + 0)(y_0 + y_3) \\ q_3 &= (x_3 + 0 + 0)(y_1 + y_2) \end{aligned} \tag{3.7}$$

A *Q/B gate* of a function F can be implemented as (3.8) where $f_F^{\langle k \rangle}(x)$ is an equation from (3.6) and x is 1-of-4 argument.

$$\begin{aligned} q_0 &= f_F^{\langle 0 \rangle}(x) \\ q_1 &= f_F^{\langle 1 \rangle}(x) \end{aligned} \tag{3.8}$$

Equations (3.4) and (3.8) in their turn imply a function for a mixer component (3.9) for two dual-rail signals $a = \{a_1, a_0\}$ and $b = \{b_1, b_0\}$.

$$q_i = a_{i_1} \cdot b_{i_0} \tag{3.9}$$

Mixers and *splitters* have a rather straight-forward positive logic implementations derived from (3.9) and (3.5) respectively. The splitter is the only component having two switching output wires.

Dual-rail gates also use positive logic decomposition. Negative logic optimisations [67] potentially can be performed, but require more sophisticated analysis of the circuit structure and are not considered within this article.

It is known that *inversion* in dual-rail can be done using "wire-crossing" [67], and inverters do not affect complexity of the conversion result. Inversion in 1-of-4 is similar, but mutual independence of operandwise signals requires separate inversions for each of them. Half inversions, when only one bit of a pair is inverted, should be defined as shown in Table 3.2.

As can be seen from the specification, described mixed radix components exhibit early propagation [80], i.e. are weakly indicating. Since m-of-n codes imply input complete gates, a completion detection mechanism is required [65, 37]. However,

Table 3.2: Inversion in 1-of-4

	full inversion	half inversion	
		lower bit	higher bit
$x_0 =$	x_3	x_1	x_2
$x_1 =$	x_2	x_0	x_3
$x_2 =$	x_1	x_3	x_0
$x_3 =$	x_0	x_2	x_1

completion detection in heterogeneous circuits is a different issue. We restrict ourselves with early propagation gates leaving other forms of speed independent solutions outside the scope of the article.

The full set of library items contains all possible 1-of-4 operandwise groupings (9 components), dual-rail and Q/B gates for basic binary functions and signal converters.

3.2.6 Benchmark results

The presented grouping algorithms have been implemented as a plug-in for Workcraft visualisation and verification environment [51, 49].

To date as far as the authors are aware no security based circuits have been proposed which use an encoding other than dual-rail, thus making them the logical choice for benchmarks. A number of arithmetic based examples were added to give a wider range for comparison. A component implementation is still a subject of improvement, therefore these tests were made for the purpose of estimation of the approach, and they do not represent final results. Since during a test we compare two circuits designed using the same library, the choice of the library doesn't make a big difference. AMS 0.35μm library was applied because of the availability of technical information.

Verilog simulation Converted circuits were compared with pure dual-rail equivalents using gate level Verilog simulation in synchronous mode. Initial circuits were mapped using complex gates as specified in (3.7). The results are shown in Table 3.3. Switching activity was measured per RTZ protocol period. In order to discover data

Table 3.3: Single rail circuits converted to dual-rail

Circuit	Dual-rail		
	Cells by radix 2	Switching wires	Sw. en., pJ
2-bit adder	10	20	11.13
16-bit ripple carry adder	80	160	86.31
4-bit multiplier*	28	56	30.08
Kasumi S-box 7	125	250	139.13
Kasumi S-box 9	128	256	146.51
Kasumi S-box 9*	150	300	169.56
AES S-box*	797	1594	818.56

* original single-rail circuits were optimised in Synopsys.

dependent variations as a result of imperfect balancing the switching energy was calculated separately for each gate output as a sum of documented values; a standard deviation of switching energy values was calculated for each benchmark circuit.

The 16-bit full adder was ideally converted (78 of 80 gates are grouped into 1-of-4) but still has only 12% power savings due to the implementation of 1-of-4 gates. From (3.4), (3.8) and (3.9) one can conclude that in fact any operandwise quaternary group consists of two Q/B gates and a mixer, and there are always two switching wires inside a 1-of-4 gate. On the other hand, Kasumi S-boxes [8] have more switching activity in 1-of-4 than in dual-rail, but consume approximately the same power. The reason is that the AMS implementation of 1-of-4 XOR-XOR component uses OA22 complex gate, when dual-rail XOR uses AO22 which consumes more power.

Original single-rail circuits for 4-bit multiplier and two S-boxes were synthesised using the complete design flow from Synopsys toolkit; in particular area and power optimisations were applied leading to negative logic decomposition. The results show these optimisations cause worse conversion results and should be taken into account when developing a design flow.

However, the switching energy of gates has minor impact on the total energy consumption of the circuits, thus the number of switching wires is of greater importance. Consequently the most significant reason of extra power cost is a large number of

Table 3.4: Mixed radix conversion results

Circuit	1-of-4 and dual-rail mixed					
	Cells by radix				Switching wires	Sw. en., pJ
	4	2	$4 \rightarrow 2$	conv.		
2-bit adder	4	0	2	1	14	10.66
16-bit ripple carry adder	39	0	2	1	84	75.72
4-bit multiplier*	10	4	4	11	58	38.72
Kasumi S-box 7	39	44	4	45	264	144.98
Kasumi S-box 9	34	59	1	38	264	137.12
Kasumi S-box 9*	44	53	9	57	326	187.96
AES S-box*	252	274	19	275	1640	1116.03

* original single-rail circuits were optimised in Synopsys.

mixers in the converted circuits that in certain cases exceeds the number of quaternary gates.

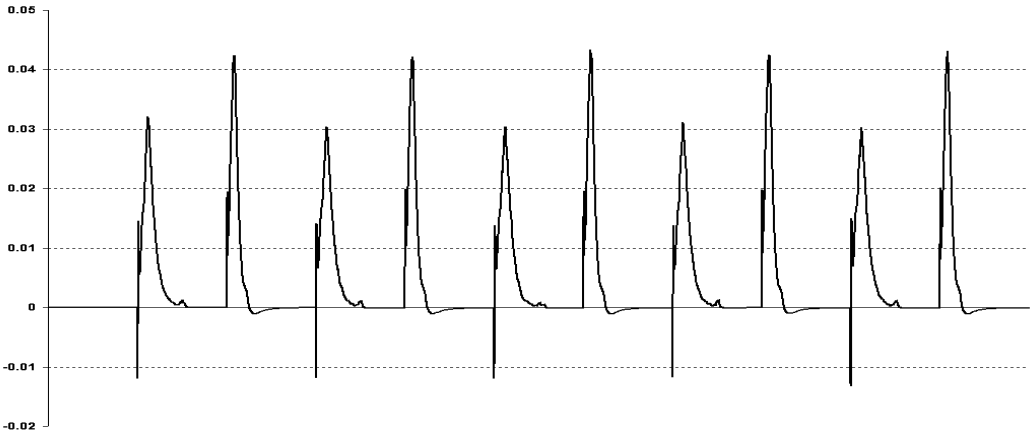
SPICE simulation AES S-box [9] implementations in single-rail, dual-rail and mixed radix were simulated in Synopsys. During the test the circuits were fed with random data inputs producing variations in the supply current over time shown in Figure 3.11. The secure implementations show immense power overheads, so the power balancing must add significant level of protection in order to justify this. The proper analysis of secure properties can be done [76]. However, as it is stated in Chapter 1, the presented work is not intended to analyse this kind of properties and refers to the previous research. From the simple visual analysis of the graphs, one can observe similar shapes of the current peaks for the dual-rail and mixed radix circuits, as they are power balanced unlike the single-rail one.

3.3 Summary

Two design flow approaches has been proposed: conversion driven design (CDD) and pre-synthesis. The first approach is based on reusing the previously made insecure designs to produce secure circuits; the flow uses Elastix tool for desynchronisation. The second approach suggests using MVL synthesis to produce encoded data path

(a) single-rail

(b) dual-rail



(c) 1-of-4

Figure 3.11: SPICE simulation results for AES S-box showing supply current, mA.

and then merge it with asynchronous control path, concurrently designed using TiDE design environment.

The method of conversion of a raw binary netlist into higher radix has been described for the case of binary to quaternary conversion. Since there are cases where the perfect conversion is impossible, the mixed radix approach is applied, and the final netlist becomes partially binary and partially quaternary. For the proposed algorithm and developed library a number of benchmarks were tested. The tests revealed power and area overheads due to the highly partitioned radix "parts" connected by signal conversion logic.

The encountered inefficiency of the CDD approach is related to a premature optimisation done by the design compiler on the netlist, before it has been passed to the conversion tool. This gives us further motivation for the synthesis approach. As in the latter approach the higher radix is taken into account at all stages of the synthesis flow, the multi-valued logic is more likely to be properly optimised and technology mapped. The following chapters are dedicated to the synthesis-based MVL design flow, which is considered as the main contribution of the presented work.

The benchmarks presented in this chapter will give the baseline for comparison between the approaches.

This chapter is based on the previously published papers [57, 55, 54].

Chapter 4

Mixed Radix Reed-Muller Expansions

Chapter 2 introduced uniform radix Reed-Muller expansions, i.e. the expansions computed entirely in one radix. This chapter presents a method of using the properties of field homomorphism, including the homomorphism under certain operations, to produce Reed-Muller expansions that can be mapped into mixed radix circuits.

4.1 Two-level mixed radix model

Reed-Muller expansions produce well-structured circuits, as seen in Figures 2.11 and 2.12.This gives the opportunity to do a better analysis of radix efficiency.

At the component level we cannot choose the globally efficient radix. Consider a simple example. Let quaternary values x, y, f be represented as pairs of binary values:$\langle x \rangle_4 = \langle x_H, x_L \rangle_2$, $\langle y \rangle_4 = \langle y_H, y_L \rangle_2$, and $\langle f \rangle_4 = \langle f_H, f_L \rangle_2$. For the function

$$f(x, y) = x \cdot y \quad \text{over GF(4)},$$

the binary representation will give

$$\begin{aligned} f_L(x_H, x_L, y_H, y_L) &= x_L y_L + x_H y_H \\ f_H(x_H, x_L, y_H, y_L) &= x_H y_L + x_L y_H + x_H y_H \end{aligned} \quad \text{over GF(2).}$$

Hence, one quaternary operation will be mapped into 8 binary. However, two binary functions

$$\begin{aligned} f_L(x_H, x_L, y_H, y_L) &= x_L y_L \\ f_H(x_H, x_L, y_H, y_L) &= x_H y_H \end{aligned} \quad \text{over GF(2)}$$

will map into the following quaternary expression:

$$f(x, y) = Bx^2y + Bxy^2 + x^2y^2 \quad \text{over GF(4)}$$

giving 2 quaternary additions and 5 quaternary multiplications.

The above example shows that the radix efficiency is different for different functions. It is also highly possible that the optimal solution might imply a combination of radices rather than a uniform radix representation. Applying the mixed radix approach at the synthesis stage will greatly extend the possibility space and the possibility of finding better solutions. The drawback of extending the search space is the increased computation time. It would be convenient to have a method to cogitatively narrow down the search space while still working with mixed radices.

Another problem related to the radix choice is in the technology level component parameters. For example, the following quaternary function

$$f = x + y \quad \text{over GF(4)}$$

in the binary will give

$$\begin{aligned} f_L &= x_L + y_L \\ f_H &= x_H + y_H \end{aligned} \quad \text{over GF(2).}$$

Quaternary representation has one operation instead of two. But as we know from the

previous chapters, quaternary components can be larger than binary, so it is not clear which of the above functions is more efficient.

At the technology level, the main reason for using different radices within a circuit is the different component costs. With respect to the GF arithmetic this means that the multiplication can be more efficient in lower radices than in higher while the addition is better in high radix implementation (or vice versa). Based on the component implementation parameters, one could use only two radices to achieve the optimal efficiency: one radix is applied to the operations of addition, another is applied to multiplication. Unlike in the conversion driven design, the synthesis implies the choice of the radix to be made beforehand. With respect to the mixed radix approach this leads to the choice of the *radix model*, i.e. the schema of applying radices at the level of mathematical representation.

For sum of product, the two-radix approach becomes a *two-level radix model*, denoted as $q_1 \rightarrow q_2$. The radix model is chosen before the synthesis stage, and the circuit is produced with respect to the radix model from the very beginning. At the level of mathematical representation, an n-variable function that complies to the two-level radix model is

$$f : GF^n(q_1) \rightarrow GF(q_2).$$

Such a function is also called *radix* q_2 *function of radix* q_1 *arguments* (or q_1-*nary-to-*q_2-*nary function*) and denoted as $f_{q_1 \rightarrow q_2}(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

Uniform radix functions are further denoted as $f_{q \rightarrow q}(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

Radix models presented in the following sections include:

- Radix reduction model $q_1 \rightarrow q_2$ for $q_1 > q_2$ has been developed for the case $p^t \rightarrow p^u$, where p is prime and $t > u$. The model is described in Section 4.2.

- Radix extension model $q_1 \rightarrow q_2$ for $q_1 < q_2$ has been developed for the cases $p \rightarrow p^t$ and $2 \rightarrow q$, where p is prime, $t > 1$ and q is any valid GF order. The models are described in Sections 4.3 and 4.5 respectively.

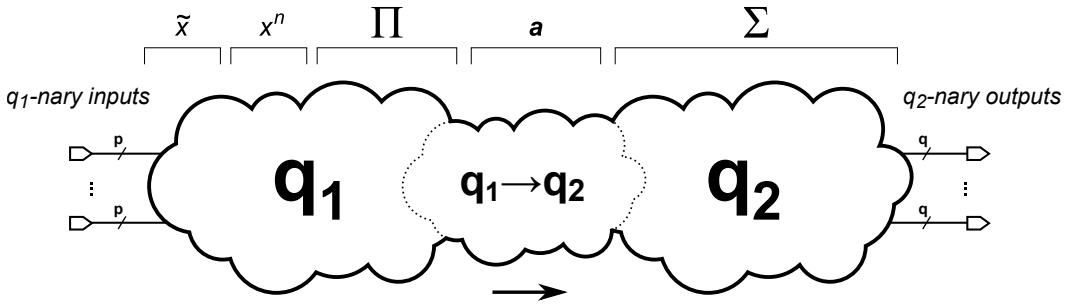


Figure 4.1: Mixed radix circuit based on $q_1 \rightarrow q_2$ two-level radix model.

We have also taken into consideration that two-level radix model might be less efficient than local per-component radix application, like in the conversion driven approach. Hence the model with mixed radix domain has been developed in order to investigate this assumption. This model is developed from $p \rightarrow p^t$ model and allows combination of radix p and radix p^t input variables within one Reed-Muller expansion. It is denoted as $\{p, p^t\} \rightarrow p^t$, where p is prime and $t > 1$, and presented in Section 4.4.

The understanding of the circuit synthesised using two-radix model approach is shown in Figure 4.1. Connection between the radices is possible without the use of signal converters due to the cross-radix arithmetic operations. In different radix models these operations are different. For example, in $p \rightarrow p^t$ radix model, multiplication of p-nary values by radix p^t coefficients will give the result of radix p^t.

4.2 Radix reduction: $p^t \rightarrow p^u, t > u$ Reed-Muller expansions

Radix reduction model is targeted at higher radix multiplications and lower radix additions in the circuit. But before we define this type of RM expansions, consider a quaternary-to-binary example using the signal conversion approach.

Considering a quaternary value $x \in GF(4)$ as a pair of binary "bits" and having defined functions $r_H(x), r_L(x) \in GF(2)$ extracting the higher and the lower bits of x respectively, any quaternary function $f_{4\rightarrow4}(\widetilde{x}_1, \ldots, \widetilde{x}_n) = a_0 + a_1\widetilde{x}_1 + \ldots + a_{4^n-1}\widetilde{x}_1^3 \cdot \ldots \cdot$

$\widetilde{x}_n^3$ can be replaced with a pair of quaternary-to-binary functions:

$$\begin{aligned} f_{H,4\to2}(\widetilde{x}_1,\ldots,\widetilde{x}_n) &= r_H(a_0) + r_H(a_1\widetilde{x}_1) + \ldots + r_H\left(a_{4^n-1}\widetilde{x}_1^3 \cdot \ldots \cdot \widetilde{x}_n^3\right) \\ f_{L,4\to2}(\widetilde{x}_1,\ldots,\widetilde{x}_n) &= r_L(a_0) + r_L(a_1\widetilde{x}_1) + \ldots + r_L\left(a_{4^n-1}\widetilde{x}_1^3 \cdot \ldots \cdot \widetilde{x}_n^3\right) \end{aligned}$$

computed over GF(2). Here functions r_H and r_L split the quaternary terms and the sum is computed over GF(2) arithmetic. In this case the total number of operations is increased: two binary additions replace each quaternary.

Example 4.1. Considering the quaternary function from Example 2.2, we can split it using the above method into the following quaternary-to-binary functions:

$$\begin{aligned} F_{H,4\to2}(x_0,x_1) &= r_H(\ddot{x_0}) + r_H\left(\ddot{x_0}^2\right) + r_H(Ax_1) + r_H\left(A\ddot{x_0}^3x_1\right) + r_H\left(x_1^2\right) + r_H\left(\ddot{x_0}^3x_1^2\right) \\ F_{L,4\to2}(x_0,x_1) &= r_L(\ddot{x_0}) + r_L\left(\ddot{x_0}^2\right) + r_L(Ax_1) + r_L\left(A\ddot{x_0}^3x_1\right) + r_L\left(x_1^2\right) + r_L\left(\ddot{x_0}^3x_1^2\right) \end{aligned}$$

where $\ddot{x}_0 = x_0 + A$.

As it is not well-defined how r_H and r_L are related, we cannot optimise the sum part of the functions. Thus the circuit shown in Figure 2.12 can be converted into a quaternary-to-binary having 20 components not considering the signal conversion logic.

Using the synthesis approach the number of operations can be optimised. However, in order to reduce radix in the middle of (2.1) we still need to introduce an explicit radix mapping defined as follows.

Definition 4.1. For some prime p and $t > u \geqslant 1$, $p^t \to p^u$ *radix mapping* is a function $r_{p^u} : GF(p^t) \to GF(p^u)$ defined in the algebra of polynomials in the transcendental variable α as (4.1). (In Chapter 2 the transcendental variable is denoted as x. To avoid confusion with function variables, in this section we denote it as α.)

$$\begin{aligned} r_{p^u}\left(a_{t-1}\alpha^{t-1} + \ldots + a_u\alpha^u + a_{u-1}\alpha^{u-1} + \ldots + a_1\alpha + a_0\right) &= \\ \alpha_{u-1}x^{u-1} + \ldots + a_1\alpha + a_0 \quad \text{over GF}(p) & \end{aligned} \tag{4.1}$$

We also use radix mapping r_{p^u} in matrix notation: $r_{p^u}(X) = Y$ for some n_1-by-n_2 matrices X, Y means that $y_{i,j} = r_{p^u}\left(x_{i,j}\right)$, where $1 \leqslant i \leqslant n_1$, $1 \leqslant j \leqslant n_2$.

From Definition 4.1 also follows that for any $x \in GF(p^t)$, $r_u(x) \in GF(p^u)$ and also $r_u(x) \in GF(p^t)$.

For example, for quaternary-to-binary radix mapping, $t = 2, u = 1$, and in the algebra of polynomials in the transcendental α,

$$r_2(a_1\alpha + a_0) = a_0 \quad \text{over GF(2)},$$

which can be implemented as

$$r_2(x) = x^3(x + A)^3 = Ax^2 + Bx \quad \text{over GF(4)}. \tag{4.2}$$

Radix mapping ensures that for each element of $GF(p^u)$ there is at least one corresponding element from $GF(p^t)$, i.e. for any $x \in GF(p^u)$ there exist $y \in GF(p^t)$, such that $r_{p^u}(y) = x$.

Corollary 3. *For $x, y \in GF(p^t)$ such that $x = r_{p^u}(y)$, the statement $r_{p^u}(x) = r_{p^u}(y)$ is also true.*

Theorem 4.1. *For some $p^t \to p^u$ radix mapping defined as in Definition 4.1, $r_{p^u}(x + y) = r_{p^u}(x) + r_{p^u}(y)$ for all $x, y \in GF(p^t)$.*

Proof. As elements of $GF(p^t)$, variables x and y are congruent to polynomials of degree $t-1$ over $GF(p)$, i.e. $x = a_{t-1}\alpha^{t-1} + \ldots + a_1\alpha + a_0$ and $y = b_{t-1}\alpha^{t-1} + \ldots + b_1\alpha + b_0$ respectively. Then

$$x + y = (a_{t-1} + b_{t-1})\,\alpha^{t-1} + \ldots + (a_1 + b_1)\,\alpha + (a_0 + b_0),$$

and therefore

$$\begin{aligned} r_{p^u}(x+y) &= (a_{u-1}+b_{u-1})\,\alpha^{u-1}+\ldots+(a_1+b_1)\,\alpha+(a_0+b_0) \\ &= \left(a_{u-1}\alpha^{u-1}+\ldots+a_1\alpha+a_0\right)+\left(b_{u-1}\alpha^{u-1}+\ldots+b_1\alpha+b_0\right) \\ &= r_{p^u}(x)+r_{p^u}(y)\,. \end{aligned}$$

□

Thus for (4.2),

$$\begin{aligned} r_2(x+y) &= A(x+y)^2+B(x+y) \\ &= Ax^2+Ay^2+Bx+By \\ &= \left(Ax^2+Bx\right)+\left(Ay^2+By\right) \\ &= r_2(x)+r_2(y)\,, \end{aligned}$$

computed over GF(4).

We define $p^t \to p^u$ RM expansion as a RM expansion over GF(p^t) where each term is mapped into GF(p^u) using radix mapping r_{p^u}, so the expansion takes the form (4.3).

$$f_{p^t\to p^u}(\widetilde{x}_1,\ldots,\widetilde{x}_n) = \sum_{i=0}^{(p^t)^n-1} r_{p^u}\left(b_i\prod_{j=1}^{n}\widetilde{x}_j^{i_j}\right) \tag{4.3}$$

where i_j is a single digit of a radix p^t tuple $\langle i_n\ldots i_1\rangle_{p^t} = i$. The argument of r_{p^u} is computed over GF(p^t) and the rest of the expression is computed over GF(p^u). The coefficients $b_0,\ldots,b_{(p^t)^n-1} \in$ GF(p^t).

From Theorem 4.1, the whole expression mapped to the radix p^u is equivalent to the expression with each term mapped to p^u, i.e. (4.3) is equivalent to the following:

$$f_{p^t\to p^u}(\widetilde{x}_1,\ldots,\widetilde{x}_n) = r_{p^u}\left(\sum_{i=0}^{(p^t)^n-1} b_i\prod_{j=1}^{n}\widetilde{x}_j^{i_j}\right),$$

which is in matrix form:

$$f_{p^t \to p^u}(\widetilde{x}_1, \ldots, \widetilde{x}_n) = r_{p^u}\left(\left(\widetilde{X}_n \otimes \ldots \otimes \widetilde{X}_1\right) \cdot b\right) \quad \text{over GF}(p^t), \tag{4.4}$$

where $\widetilde{X}_j = \begin{bmatrix} 1 & \widetilde{x}_j & \ldots & \widetilde{x}_j^{(p^t)^n-1} \end{bmatrix}$ and $b = \begin{bmatrix} b_0 & \ldots & b_{(p^t)^n-1} \end{bmatrix}^t$.

Having specified the truth vector $d = \begin{bmatrix} d_0 & \ldots & d_{(p^t)^n-1} \end{bmatrix}^t$ and mapping it into radix p^u, $d' = r_{p^u}(d)$, we can write

$$r_{p^u}\left(\left(\widetilde{X}_n \otimes \ldots \otimes \widetilde{X}_1\right) \cdot b\right) = r_{p^u}(d) = d'.$$

If we find such b' that

$$\left(\widetilde{X}_n \otimes \ldots \otimes \widetilde{X}_1\right) \cdot b' = d' \quad \text{over GF}(p^t), \tag{4.5}$$

then according to Corollary 3:

$$r_{p^u}\left(\left(\widetilde{X}_n \otimes \ldots \otimes \widetilde{X}_1\right) \cdot b'\right) = d' = r_{p^u}\left(\left(\widetilde{X}_n \otimes \ldots \otimes \widetilde{X}_1\right) \cdot b\right).$$

Since r_{p^u} is not one-to-one function, the above expression is true even if $b \neq b'$. Therefore, $p^t \to p^u$ RM expansion can have more than one solution, and b' is one of them.

Following from d is constant, $d' = r_{p^u}(d)$ is constant too. Thus b' can be found from (4.5) using Green's direct method for radix p^t:

$$b' = (S_{k_n} \otimes \ldots \otimes S_{k_1})^{-1} \cdot d' = \left(S_{k_n}^{-1} \otimes \ldots \otimes S_{k_1}^{-1}\right) \cdot d' \quad \text{over GF}(p^t), \tag{4.6}$$

where the matrices $S_0, \ldots, S_{p^t-1}$ are defined for uniform radix p^t expansions as described in Section 2.2.3, and k is a polarity number.

Mapping r_{p^u} is a signal conversion function that requires logic to implement, i.e. it has nonzero cost at the physical level. Having r_{p^u} in front of each of $(p^t)^n$ terms in (4.3)

	x	$r_2(x)$	$r_2(x)+r_2(y)$	0	1	A	B
	0	0	0	0	1	0	1
GF(4):	1	1	1	1	0	1	0
	A	0	A	0	1	0	1
	B	1	B	1	0	1	0

Figure 4.2: $r_2(x)+r_2(y)$ component specification for quaternary-to-binary case

gives concerns about this radix model in general. However, instead of making $r_{p^u}(x)$ a separate component, the mapping can be merged with the consequent operation of addition into a special macro-component $r_{p^u}(x)+r_{p^u}(y)$, which has simpler truth table, e.g the one shown in Figure 4.2. It has radix p^t inputs and radix p^u and behaves similarly to Q/B gates from conversion driven design (Chapter 3) consuming signal conversion components in the circuit.

Example 4.2. In order to make the quaternary-to-binary equivalent of the function from the Example 2.2, quaternary truth vector d can be split into two binary truth vectors $d_{0,4\to2}$ and $d_{1,4\to2}$ in a way that each element of d is represented as a pair of respective elements from $d_{0,4\to2}$ and $d_{1,4\to2}$, i.e. $\langle d_i\rangle_4 = \langle d_{1,4\to2,i}, d_{0,4\to2,i}\rangle_2$ for $i = 0\ldots15$.

$$\begin{aligned} d_{0,4\to2} &= [1100111011001110]^t \\ d_{1,4\to2} &= [1111110111111101]^t \end{aligned}$$

Hence the circuit will implement two quaternary-to-binary functions $F_{0,4\to2}$ and $F_{1,4\to2}$. Let polarity $k = 2 = \langle 0A\rangle_4$. Using Green's matrices W_0 and W_2 as defined for quaternary RM expansions, we have:

$$b'_0 = (W_0 \otimes W_2)\cdot d_{0,4\to2} \quad \text{over GF(4)},$$

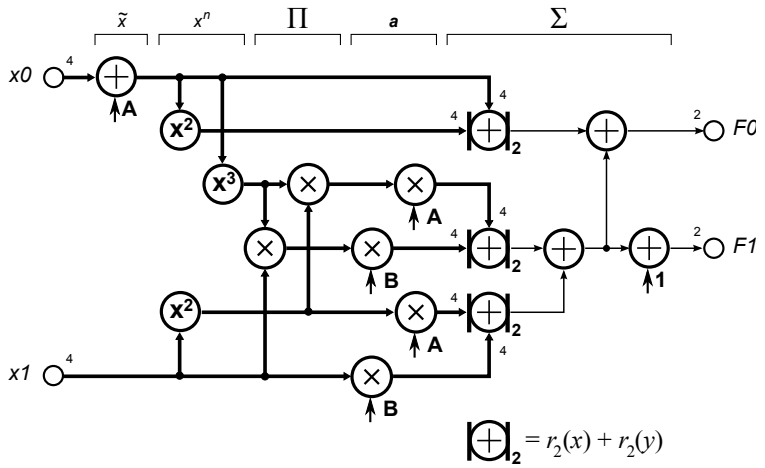


Figure 4.3: Component-level schematic for quaternary-to-binary RM expansion.

$$
\mathrm{b}_0' = \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \mathrm{B} & \mathrm{A} \\ 0 & 1 & \mathrm{A} & \mathrm{B} \\ 1 & 1 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 1 & 0 \\ \mathrm{B} & \mathrm{A} & 0 & 1 \\ \mathrm{A} & \mathrm{B} & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \right) \cdot \mathrm{d}_{0,4\to 2} =
$$

$$
= \begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathrm{B} & \mathrm{A} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathrm{A} & \mathrm{B} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \mathrm{B} & 0 & 0 & 0 & \mathrm{A} & 0 \\
0 & 0 & 0 & 0 & \mathrm{B} & \mathrm{A} & 0 & 1 & \mathrm{A} & 1 & 0 & \mathrm{B} & 1 & \mathrm{B} & 0 & \mathrm{A} \\
0 & 0 & 0 & 0 & \mathrm{A} & \mathrm{B} & 0 & 1 & 1 & \mathrm{A} & 0 & \mathrm{B} & \mathrm{B} & 1 & 0 & \mathrm{A} \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \mathrm{B} & \mathrm{B} & \mathrm{B} & \mathrm{B} & \mathrm{A} & \mathrm{A} & \mathrm{A} & \mathrm{A} \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \mathrm{A} & 0 & 0 & 0 & \mathrm{B} & 0 \\
0 & 0 & 0 & 0 & \mathrm{B} & \mathrm{A} & 0 & 1 & 1 & \mathrm{B} & 0 & \mathrm{A} & \mathrm{A} & 1 & 0 & \mathrm{B} \\
0 & 0 & 0 & 0 & \mathrm{A} & \mathrm{B} & 0 & 1 & \mathrm{B} & 1 & 0 & \mathrm{A} & 1 & \mathrm{A} & 0 & \mathrm{B} \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \mathrm{A} & \mathrm{A} & \mathrm{A} & \mathrm{A} & \mathrm{B} & \mathrm{B} & \mathrm{B} & \mathrm{B} \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\mathrm{B} & \mathrm{A} & 0 & 1 & \mathrm{B} & \mathrm{A} & 0 & 1 & \mathrm{B} & \mathrm{A} & 0 & 1 & \mathrm{B} & \mathrm{A} & 0 & 1 \\
\mathrm{A} & \mathrm{B} & 0 & 1 & \mathrm{A} & \mathrm{B} & 0 & 1 & \mathrm{A} & \mathrm{B} & 0 & 1 & \mathrm{A} & \mathrm{B} & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \mathrm{B} \\ 0 \\ 0 \\ \mathrm{B} \\ \mathrm{A} \\ 0 \\ 0 \\ \mathrm{A} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.
$$

Similarly for $\mathrm{F}_{0,4\to 2}$, we have $\mathrm{b}_1' = (W_0 \otimes W_2) \cdot \mathrm{d}_{1,4\to 2} = [1000\mathrm{B}00\mathrm{BA}00\mathrm{A}0000]^{\mathrm{t}}$. Hence

the functions take the following form:

$$\begin{aligned} F_{0,4\to2}(x_0,x_1) &= r_2(\ddot{x_0}) + r_2\left(\ddot{x_0}^2\right) + r_2(Bx_1) + r_2\left(Bx_0^3x_1\right) + r_2\left(Ax_1^2\right) + r_2\left(Ax_0^3x_1^2\right) \\ F_{1,4\to2}(x_0,x_1) &= 1 + r_2(Bx_1) + r_2\left(Bx_0^3x_1\right) + r_2\left(Ax_1^2\right) + r_2\left(Ax_0^3x_1^2\right), \end{aligned}$$

where $\ddot{x_0} = x_0 + A$.

Although the functions have many operations, most of the terms are shared, so the final circuit is reduced in size. Figure 4.3 shows the schematic of the circuit using $r_2(x) + r_2(y)$ components.

Comparing the mixed radix examples with Example 2.1 and 2.2, one could conclude that mixed radix circuits give better area and possibly reduce power consumption, however this is not necessarily true. Component-level examples illustrate the potential of the approach regardless of the implementation, while at the physical level the arithmetic operations have different costs in different radices depending on the representation of multi-valued signals. Unfortunately, most of the theoretical papers do not cover this fact. In our work we have considered a specific application area and esimated the real characteristics presented in Chapter 5.

4.3 Radix extension: $p \to p^t$ Reed-Muller expansions

From Definition 2.8, GF(p) is a subfield of GF(p^t). Hence, the elements of GF(p) are a subset of GF(p^t), and the operations over these elements are equivalent in both fields:

$$\begin{aligned} x + y \text{ over GF}(p) &= x + y \text{ over GF}(p^t) \\ x \cdot y \text{ over GF}(p) &= x \cdot y \text{ over GF}(p^t) \end{aligned}$$

for all $x, y \in$GF(p).

We define $p \to p^t$ expansion as a Reed-Muller expansion $f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ over GF(p^t) with arguments constrained to the elements of GF(p), i.e $\widetilde{x}_j \in$GF(p) and $x_j \in$GF(p) for

$j = 1 \ldots n$. Let us find its generalised form and prove that all operations in the product part can be computed over GF(p) while the sum part is computed over GF(p^t).

Since there is a constraint on variables $x_1 \ldots x_n$ and their literals $\widetilde{x}_1 \ldots \widetilde{x}_n$, there are only p polarity forms allowed: $\widetilde{x}_j = x_j + c, j = 1 \ldots n$, where $x_j, c \in$GF(p). The operation of addition here is respectively computed over GF(p).

Consider one-variable RM expansion, which for radix p^t function $f(x)$ takes the form:

$$f_{p^t \to p^t}(\widetilde{x}) = a_0 + a_1\widetilde{x} + \ldots + a_{p^t-1}\widetilde{x}^{p^t-1} \quad \text{over GF}(p^t) \tag{4.7}$$

Constraining variable x and literal $\widetilde{x}$ to the values of GF(p), we have $\widetilde{x} = \widetilde{x}^p$, $x \in$GF(p), $\widetilde{x} \in$GF(p). Following from the basic Galois theory, the number of different power forms is reduced to p, and the result is also in GF(p) [52]:

$$\begin{aligned} \widetilde{x}^p &= \widetilde{x} \\ \widetilde{x}^p \cdot \widetilde{x} &= \widetilde{x}^2 \\ &\ldots \\ \widetilde{x}^{p^t-1} &= \widetilde{x}^{p-1} \end{aligned}$$

Or generally,

$$\widetilde{x}^i = \widetilde{x}^{\psi(i)} \quad \text{over GF}(p^t)$$

for $i = 0 \ldots (p^t - 1)$, where

$$\psi(i) = \begin{cases} 0 & \text{if } i = 0 \\ [(i-1) \bmod (p-1)] + 1 & \text{if } i > 0 \end{cases}$$

The function ψ looks convoluted, but it describes a rather simple pattern. For example, in case of GF($3^2 = 9$), we have $\psi(0) = 0, \psi(1) = \psi(3) = \ldots = 1, \psi(2) = \psi(4) = \ldots = 2$, and the powers of $\widetilde{x}$ will be:

$\widetilde{x}^0$	$\widetilde{x}$	$\widetilde{x}^2$	$\widetilde{x}^3$	$\widetilde{x}^4$	$\widetilde{x}^5$	$\widetilde{x}^6$	$\widetilde{x}^7$	$\widetilde{x}^8$
1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1
1	2	1	2	1	2	1	2	1

And in the case of $p = 2$, the function ψ becomes trivial:

$$\psi(i) = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i > 0 \end{cases}$$

Therefore (4.7) can be rewritten as (4.8) giving us the form of one-variable $p \rightarrow p^t$ RM expansion.

$$\begin{aligned} f_{p\rightarrow p^t}(\widetilde{x}) &= a_0 + \left(a_1 + a_p + \ldots + a_{p^t-p+1}\right)\widetilde{x} + \ldots + \left(a_{p-1} + \ldots + a_{p^t-1}\right)\widetilde{x}^{p-1} \\ &= b_0 + b_1\widetilde{x} + \ldots + b_{p-1}\widetilde{x}^{p-1} \quad \text{over GF}(p^t) \qquad (4.8) \\ b_0 &= a_0 \\ b_1 &= a_1 + a_p + \ldots + a_{p^t-p+1} \\ &\ldots \\ b_{p-1} &= a_{p-1} + \ldots + a_{p^t-1} \end{aligned}$$

Radix p^t expansion of n-variable function, according to (2.1), is expressed as follows.

$$f_{p^t\rightarrow p^t}(\widetilde{x}_1, \ldots, \widetilde{x}_n) = \sum_{i=0}^{(p^t)^n-1} a_i \left[\prod_{j=1}^{n} \widetilde{x}_j^{i_j}\right] \quad \text{over GF}(p^t), \qquad (4.9)$$

where i_j is a single digit of a radix p^t tuple $\langle i_n \ldots i_1 \rangle_{p^t} = i$. Vector $a = \begin{bmatrix} a_0 & \ldots & a_{(p^t)^n-1} \end{bmatrix}^t$ is a radix p^t coefficient vector.

For an n-variable case the same idea can be used since $\widetilde{x}_j = \widetilde{x}_j^p$ is true for all literals $\widetilde{x}_j \in$GF(p), $j = 1 \ldots n$; thus each term in (4.9) is transformed as follows.

$$a_i \widetilde{x}_1^{i_1} \ldots \widetilde{x}_n^{i_n} = a_i \widetilde{x}_1^{\psi(i_1)} \ldots \widetilde{x}_n^{\psi(i_n)}$$

For example, in the binary-to-quaternary RM expansion ($p = 2$, $p^t = 4$), a quaternary term $a_{19}\widetilde{x}_1^3\widetilde{x}_3$ will become $a_{19}\widetilde{x}_1\widetilde{x}_3$, and the term $a_{36}\widetilde{x}_1\widetilde{x}_3^2$ will also give $a_{36}\widetilde{x}_1\widetilde{x}_3$. As the reader may notice, the multiple terms now have common multipliers. Using factorisation, $\left(p^t\right)^n$ terms of (4.9) (n variables, each having p^t different power forms) is reduced to p^n as we now have only p possible powers.

Finally, the fixed polarity Reed-Muller expansion over GF(p^t) of the radix p^t function $f(x_1,\ldots,x_n)$ takes the form (4.10) if each of the variables $x_j \in$GF(p) and $\widetilde{x}_j \in$GF(p); GF(p) $\subset$ GF(p^t).

$$f_{p\to p^t}(\widetilde{x}_1,\ldots,\widetilde{x}_n) = \sum_{i=0}^{p^n-1} b_i \left[\prod_{j=1}^{n} \widetilde{x}_j^{i_j}\right] \quad \text{over GF}(p^t) \tag{4.10}$$

where i is an integer representation of a p-nary tuple $\langle i_n \ldots i_1\rangle_p = i$, and vector $b = \left[\begin{array}{ccc} b_0 & \ldots & b_{p^n-1}\end{array}\right]^t$ is a coefficient vector, which can be derived from coefficient vector a as follows:

$$\begin{aligned}
b_i &= \sum_{m=0}^{(p^t)^n-1} \omega_{i,m} a_m, i = 0 \ldots p^n - 1 \\
\omega_{i,m} &= \begin{cases} 1 & \text{if there exists such } l \text{that } i_l = \psi(m_l) \\ 0 & \text{otherwise} \end{cases} \\
\langle i\rangle_{10} &= \langle i_n \ldots i_1\rangle_p \\
\langle m\rangle_{10} &= \langle m_n \ldots m_1\rangle_{p^t}
\end{aligned}$$

Here the factor $\omega_{i,m}$ is used to select coefficients from the terms with common multipliers; $b_i \in$GF(p^t) because $a_m \in$GF(p^t).

The difference between (4.10) and (4.9) is that the $p \to p^t$ expansion has p^n terms and a p-nary representation of $\langle i\rangle_{10} = \langle i_n \ldots i_1\rangle_p$, i.e. it has a form of a *radix* p RM expansion computed over GF(p^t). It is also important that, despite the entire expression is being computed over GF(p^t), the computation of the product part remains within GF(p).

Finding coefficient vector b from the coefficients $a_0 \dots a_{q^n-1}$ described above is given as a proof of the fact that mixed radix $p \rightarrow p^t$ expansions can be derived from the uniform radix, but it has no practical use, as the coefficient vector a is usually unknown. Instead, the coefficient vector b can be found from the truth vector in a similar manner as Green's direct method described in Section 2.2.3.

Since each literal $\widetilde{x}_j \in GF(p)$ has only p power forms, we have $\widetilde{X}_j = \begin{bmatrix} 1 & \widetilde{x}_j & \dots & \widetilde{x}_j^{p-1} \end{bmatrix}$ similar to the radix p case of RM expansions; $j = 1 \dots n$. The matrix form of (4.10) is:

$$f_{p \rightarrow p^t}(\widetilde{x}_1, \dots, \widetilde{x}_n) = \left(\widetilde{X}_n \otimes \dots \otimes \widetilde{X}_1\right) \cdot b \quad \text{over } GF(p^t). \tag{4.11}$$

Let $d = \begin{bmatrix} d_0 & d_1 & \dots & d_{p-1} \end{bmatrix}^t$ be the truth vector of the 1-variable $p \rightarrow p^t$ RM expansion (4.8); $d_0, d_1, \dots, d_{p-1} \in GF(p^t)$. Then, for some polarity number k, from (4.11) follows:

$$d = S_k b \quad \text{over } GF(p^t),$$

where S_k is p-by-p matrix as defined for p-nary RM expansions. Having $W_k = S_k^{-1}$, the equation can be solved giving the coefficient vector b:

$$b = W_k d \quad \text{over } GF(p^t).$$

The elements of matrix W_k have radix p. The multiplication with the radix p^t truth vector d produces coefficient vector b of radix p^t as required.

In general n-variable case, for radix p^t truth vector d, radix p matrices $S_0, \dots, S_{p-1}$ and $W_0 = S_0^{-1}, \dots, W_{p-1} = S_{p-1}^{-1}$, the truth vector b can be found as follows:

$$b = (S_{k_n} \otimes \dots \otimes S_{k_1})^{-1} \cdot d \quad \text{over } GF(4)$$

$$\begin{aligned}(S_{k_n} \otimes \ldots \otimes S_{k_1})^{-1} &= S_{k_n}^{-1} \otimes \ldots \otimes S_{k_1}^{-1} \\ &= W_{k_n} \otimes \ldots \otimes W_{k_1}.\end{aligned}$$

The equation has a solution if the matrices $S_0, \ldots, S_{p-1}$ are invertible over GF(p), which means that if Green's method can be applied to some radix p, the radix model $p \rightarrow p^t$ is also valid for this p and can be solved using the presented method.

Example 4.3. In the binary-to-quaternary case, the truth vectors d_0 and d_1 from the Example 2.1 can be merged into quaternary $d_{2\rightarrow4} = [BBAABB1A]^t$.

For $k = 1 = \langle 001 \rangle_2$ and using binary case matrices W_0 and W_1, we can find the coefficient vector b:

$$b = (W_0 \otimes W_0 \otimes W_1) \cdot d_{2\rightarrow4} \quad \text{over GF(4)},$$

$$\begin{aligned} b &= \left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \right) \cdot d_{2\rightarrow4} \\ &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} B \\ B \\ A \\ A \\ B \\ B \\ 1 \\ A \end{bmatrix} = \begin{bmatrix} B \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ B \end{bmatrix}. \end{aligned}$$

Hence, the 3-variable binary-to-quaternary function takes the following form:

$$F_{2\rightarrow4}(y_0, y_1, y_2) = B + y_1 + B\overline{y_0}y_1y_2$$

where $\overline{y_0} = y_0 + 1$.

The schematics are shown in Figure 4.4 displaying how the radix changes from layer to layer.

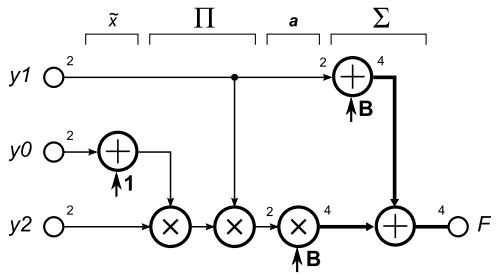


Figure 4.4: Component-level schematic for mixed radix binary-to-quaternary RM expansion.

4.4 Mixed radix domain: $\{p, p^t\} \rightarrow p^t$ Reed-Muller expansions

Uniform radix p^t expansions and radix p^t expansions of p-nary arguments are the extremes the of more general mixed radix domain expansions $\{p, p^t\} \rightarrow p^t$ allowing input from both radices within a single circuit.

Consider a 2-variable quaternary RM expansion

$$f(\widetilde{x}_1, \widetilde{x}_2) = \left(\widetilde{X}_2 \otimes \widetilde{X}_1\right) \cdot a \quad \text{over GF(4)},$$

where $\widetilde{X}_i = \begin{bmatrix} 1 & x_i & x_i^2 & x_i^3 \end{bmatrix}$, $i \in \{1,2\}$, and a 4-variable binary-to-quaternary RM expansion

$$f(\widetilde{y}_{10}, \widetilde{y}_{11}, \widetilde{y}_{20}, \widetilde{y}_{21}) = \left(\widetilde{Y}_{21} \otimes \widetilde{Y}_{20} \otimes \widetilde{Y}_{11} \otimes \widetilde{Y}_{10}\right) \cdot b \quad \text{over GF(4)},$$

where $\widetilde{Y}_{ij} = \begin{bmatrix} 1 & y_i \end{bmatrix}$, $i \in \{1,2\}$, $j \in \{0,1\}$. Denoting $\widetilde{Y}_i = \widetilde{Y}_{i1} \otimes \widetilde{Y}_{i0} = \begin{bmatrix} 1 & y_{i0} & y_{i1} & y_{i0}y_{i1} \end{bmatrix}$, we can write

$$f(\widetilde{y}_{10}, \widetilde{y}_{11}, \widetilde{y}_{20}, \widetilde{y}_{21}) = \left(\left(\widetilde{Y}_{21} \otimes \widetilde{Y}_{20}\right) \otimes \left(\widetilde{Y}_{11} \otimes \widetilde{Y}_{10}\right)\right) \cdot b = \left(\widetilde{Y}_2 \otimes \widetilde{Y}_1\right) \cdot b \quad \text{over GF(4)}.$$

In a similar manner we can deduce the following constructs:

$$f(\widetilde{y}_{10}, \widetilde{y}_{11}, \widetilde{x}_2) = \left(\widetilde{X}_2 \otimes \left(\widetilde{Y}_{11} \otimes \widetilde{Y}_{10}\right)\right) \cdot e_1 = \left(\widetilde{X}_2 \otimes \widetilde{Y}_1\right) \cdot e_1 \quad \text{over GF(4)},$$

$$f(\widetilde{x}_1, \widetilde{y}_{20}, \widetilde{y}_{21}) = \left(\left(\widetilde{Y}_{21} \otimes \widetilde{Y}_{20}\right) \otimes \widetilde{X}_1\right) \cdot e_2 = \left(\widetilde{Y}_2 \otimes \widetilde{X}_1\right) \cdot e_2 \quad \text{over GF(4)},$$

which will give us the notion of mixed radix domain functions (e_1 and e_2 are some coefficient vectors).

Mixed radix domain $\{p, p^t\} \rightarrow p^t$ Reed-Muller expansion is defined as a radix p^t function $f(\widetilde{z}_1, \ldots, \widetilde{z}_n)$, where each literal $\widetilde{z}_j$ can be either the radix p^t variable $\widetilde{x}_j$ or a p-nary tuple $\langle \widetilde{y}_{j,t-1}, \ldots, \widetilde{y}_{j,0} \rangle_p$ representing $\widetilde{x}_j$, $j = 1 \ldots n$. Such t-tuple can express the coefficients of the corresponding polynomial of transcendental variable α, so we can say that $x_j = y_{j,t-1}\alpha^{t-1} + \ldots + y_{j,1}\alpha + y_{j,0}$ over GF(p). By Definition 2.11, literal $\widetilde{x}_j = x_j + c$. Let c be expressed with a p-nary tuple as well, i.e. $c = \langle c_{t-1}, \ldots, c_0 \rangle_p$, then

$$\begin{aligned} \widetilde{x}_j = x_j + c &= \left(y_{j,t-1}\alpha^{t-1} + \ldots + y_{j,1}\alpha + y_{j,0}\right) + \left(c_{t-1}\alpha^{t-1} + \ldots + c_1\alpha + e_0\right) \\ &= \left(y_{j,t-1} + c_{t-1}\right)\alpha^{t-1} + \ldots + \left(y_{j,1} + c_1\right)\alpha + \left(y_{j,0} + c_0\right) \\ &= \left\langle y_{j,t-1} + c_{t-1}, \ldots, y_{j,0} + c_0 \right\rangle_p = \left\langle \widetilde{y}_{j,t-1}, \ldots, \widetilde{y}_{j,0} \right\rangle_p, \end{aligned}$$

where $\widetilde{y}_{j,i} = y_{j,i} + c_i$, $i = 0 \ldots t-1$. As literal $\widetilde{x}_j$ is determined by a single digit k_j of a polarity number $\langle k \rangle_{10} = \langle k_n \ldots k_1 \rangle_{p^t}$, in order to preserve the consistency between the literals $\widetilde{x}_j$ and $\widetilde{y}_{j,t-1}, \ldots, \widetilde{y}_{j,0}$, the same polarity number can be expressed as a p-nary tuple, i.e. for each $j = 1 \ldots n$ we have $\langle k_j \rangle_{p^t} = \langle k_{j,t-1} \ldots k_{j,0} \rangle_p$.

Definition 4.2. *Argument radix number* r of a mixed radix domain RM expansion $f(\widetilde{z}_1, \ldots, \widetilde{z}_n)$ is an integer representation of a binary tuple $\langle r_n \ldots r_1 \rangle_2 = r$ where r_j is 0 if $\widetilde{z}_j = \widetilde{x}_j$, or 1 if $\widetilde{z}_j = \langle \widetilde{y}_{j,t-1}, \ldots, \widetilde{y}_{j,0} \rangle_p$; $j = 1 \ldots n$. For uniform radix expansions $r = 0$; for radix p^t expansions of all p-nary arguments $r = 2^n - 1$.

For some radix number $\langle r \rangle_{10} = \langle r_n \ldots r_1 \rangle_2$, mixed radix domain $\{p, p^t\} \rightarrow p^t$ Reed-Muller expansion of n-variable radix p^t function takes the following form:

$$f(\widetilde{z}_1, \ldots, \widetilde{z}_n) = \left(\widetilde{Z}_n \otimes \ldots \otimes \widetilde{Z}_1\right) \cdot e_r \quad \text{over GF}(p^t),$$

where

$$\widetilde{Z}_j = \begin{cases} \widetilde{X}_j & \text{if } r_j = 0 \\ \widetilde{Y}_j = \widetilde{Y}_{j,t-1} \otimes \ldots \otimes \widetilde{Y}_{j,0} & \text{if } r_j = 1 \end{cases}$$
$$j = 1 \ldots n$$

considering that $\widetilde{X}_j = \begin{bmatrix} 1 & \widetilde{x}_j & \ldots & \widetilde{x}_j^{p^t-1} \end{bmatrix}$ and $\widetilde{Y}_j = \begin{bmatrix} 1 & \widetilde{y}_j & \ldots & \widetilde{y}_j^{p-1} \end{bmatrix}$.

The computation of the coefficient vector e_r is derived from the Green's direct method. Let the truth vector be defined as $d = \begin{bmatrix} d_0 & \ldots & d_{(p^t)^n-1} \end{bmatrix}^t$, $d_0 \ldots d_{(p^t)^n-1} \in GF(p^t)$. Denoting radix p matrices as $S_{p-1}, \ldots, S_0$and $W_{p-1}, \ldots, W_0$, and radix p^t matrices as $S'_{p^t-1}, \ldots, S_0$and $W'_{p^t-1}, \ldots, W'_0$, we can compute mixed radix domain RM expansion as follows:

$$d = (T_{k_n} \otimes \ldots \otimes T_{k_1}) \cdot e_r \quad \text{over } GF(p^t),$$

$$T_{k_j} = \begin{cases} S'_{k_j} & \text{if } r_j = 0 \\ S_{k_{j,t-1}} \otimes \ldots \otimes S_{k_{j,0}} & \text{if } r_j = 1 \end{cases}$$
$$j = 1 \ldots n.$$

Hence,

$$e_r = \left(T_{k_n}^{-1} \otimes \ldots \otimes T_{k_1}^{-1}\right) \cdot d \quad \text{over } GF(p^t),$$

$$T_{k_j}^{-1} = \begin{cases} W'_{k_j} & \text{if } r_j = 0 \\ W_{k_{j,t-1}} \otimes \ldots \otimes W_{k_{j,0}} & \text{if } r_j = 1 \end{cases}$$
$$j = 1 \ldots n.$$

Exhaustive search through 2^n argument radix numbers and computing for each of the 4^n fixed polarity expansions is a high complexity task. Benchmark results

presented in Section **??** show that in most cases it is unnecessary to search through all radix numbers as the optimal solution trend to either all radix p arguments or all radix p^t depending on the properties of arithmetic components and the function.

Example 4.4. Using the truth vector $d = [BBAABB1ABBAABB1A]^t$ of the two-variable quaternary function from Example 2.2.

For $r = 1 = \langle 01 \rangle_2$, which gives $\widetilde{z}_0 = (y_{01}, y_{00})$ and $\widetilde{z}_1 = \widetilde{x}_1$, and for $k = 1 = \langle 0, \langle 01 \rangle_2 \rangle_4$, the coefficient vector e_1 can be found as follows.

$$e_1 = \left(W'_0 \otimes (W_0 \otimes W_1)\right) \cdot d \quad \text{over GF(4)}$$

$$e_1 = \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & B & A \\ 0 & 1 & A & B \\ 1 & 1 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \right) \cdot d$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & B & 0 & 0 & 0 & A & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & B & B & 0 & 0 & A & A & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & B & 0 & B & 0 & A & 0 & A \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & B & B & B & B & A & A & A & A \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & A & 0 & 0 & 0 & B & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & A & A & 0 & 0 & B & B & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & A & 0 & A & 0 & B & 0 & B \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & A & A & A & A & B & B & B & B \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} B \\ B \\ A \\ A \\ B \\ B \\ 1 \\ A \\ B \\ B \\ A \\ A \\ B \\ B \\ 1 \\ A \end{bmatrix} = \begin{bmatrix} B \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ A \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Hence the mixed radix domain function $F_{\{2,4\}\to 4}$ for radix number $r = 1$ takes the form:

$$F_{\{2,4\}\to 4}(x_1, y_{01}, y_{00}) = B + y_{01} + Ax_1y_{01} + x_1^2\overline{y_{00}}y_{01},$$

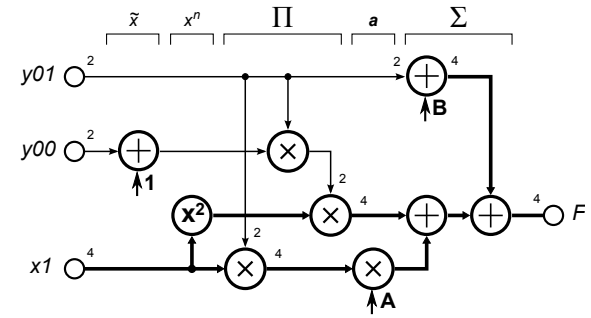
where $\overline{y_{00}} = y_{00} + 1$. The component level schematic is shown in Figure 4.5(a).

For $r = 2 = \langle 02\rangle_2$, which gives $\widetilde{z}_0 = \widetilde{x}_0$ and $\widetilde{z}_1 = (y_{11}, y_{10})$, and for $k = 2 = \langle\langle 00\rangle_2 A\rangle_4$, the coefficient vector e_2 can be found as follows.

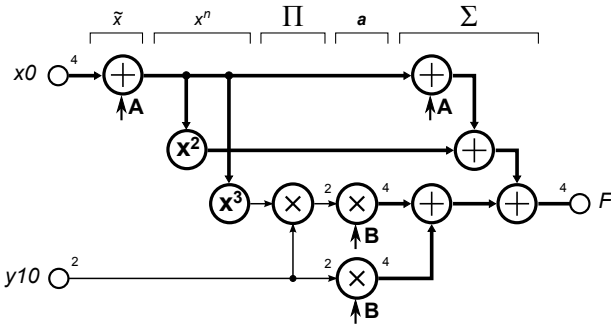
$$e_2 = \left((W_0 \otimes W_0) \otimes W_2'\right) \cdot d \quad \text{over GF(4)}$$

$$e_2 = \left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 1 & 0 \\ B & A & 0 & 1 \\ A & B & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \right) \cdot d$$

$$= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B & A & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & B & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B & A & 0 & 1 & B & A & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & B & 0 & 1 & A & B & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ B & A & 0 & 1 & 0 & 0 & 0 & 0 & B & A & 0 & 1 & 0 & 0 & 0 & 0 \\ A & B & 0 & 1 & 0 & 0 & 0 & 0 & A & B & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ B & A & 0 & 1 & B & A & 0 & 1 & B & A & 0 & 1 & B & A & 0 & 1 \\ A & B & 0 & 1 & A & B & 0 & 1 & A & B & 0 & 1 & A & B & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} B \\ B \\ A \\ A \\ B \\ B \\ 1 \\ A \\ B \\ B \\ A \\ A \\ B \\ B \\ 1 \\ A \end{bmatrix} = \begin{bmatrix} A \\ 1 \\ 1 \\ 0 \\ B \\ 0 \\ 0 \\ B \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$



(a) $r = 1$



(b) $r = 2$

Figure 4.5: Component-level schematic for mixed-to-quaternary RM expansions.

Hence the mixed radix domain function $F_{\{2,4\}\to 4}$ for radix number $r = 2$ takes the form:

$$F_{\{2,4\}\to 4}(y_{11}, y_{10}, x_0) = A + \ddot{x}_0 + \ddot{x}_0^2 + By_{10} + By_{10}\ddot{x}_0^3,$$

where $\ddot{x}_0 = \ddot{x}_0 + A$. The component level schematic is shown in Figure 4.5(b). The circuit does not have y_{11} input because the function does not actually depend on it, which can be observed from its truth vector.

4.5 Binary-to-q-nary Reed-Muller Expansions

Radix extension model described in Section 4.3 is based on the homomorphism property of field extensions. However, as we compute only the product part in the lower radix, we do not actually need the fields to be homomorphic under the operation of addition.

Since all felds contain zero element 0 and unity element 1, and also $0 \cdot 0 = 0$, $0 \cdot 1 =$

$1 \cdot 0 = 0$ and $1 \cdot 1 = 1$ in all fields, we can say that GF(2) is homomorphic to any other Galois field under the operation of multiplication.

Function $f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ is *binary-to-*q*-nary* if $f(\widetilde{x}_1, \ldots, \widetilde{x}_n) \in GF(q)$, variables $x_1, \ldots, x_n \in \{0,1\}$ and their polarity forms $\widetilde{x}_1, \ldots, \widetilde{x}_n \in \{0,1\}$; $\{0,1\} \subset GF(q)$. Such a function is also called q-*nary function of binary arguments* and denoted as $f_{2\to q}(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

For $q = 2^m$, condition $\widetilde{x}_j \in \{0,1\}$, $j = 1 \ldots n$ is true for 0 and 1 polarity forms, i.e. $\widetilde{x}_j = x_j + c_j$; $x_j, c_j \in \{0,1\}$. However, for other q this condition is not true, so for binary-to-q-nary case we have to alter Definition 2.11 as follows.

Definition 4.3. *Literal* $\widetilde{x}_j$, $j = 1 \ldots n$ of the variable $x_j \in \{0,1\}$ w.r.t. the binary-to-q-nary function $f_{2\to q}(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ is the one of two possible polarity forms:

$$\begin{aligned} x_j &= x_j + 0 \\ \overline{x}_j &= (-1)\, x_j + 1 \end{aligned}$$

where 0, 1, x_j and (-1) are elements of GF(q). The polarity forms are denoted as 0-polarity and 1-polarity respectively.

For example, in GF(3) $(-1) = 2$ giving $\overline{x}_j = 2x_j + 1$. If $x_j = 0$, $\overline{x}_j = 2 \cdot 0 + 1 = 1$; if $x_j = 1$, $\overline{x}_j = 2 \cdot 1 + 1 = 0$. Hence, $\overline{x}_j \in \{0,1\}$ if $x_j \in \{0,1\}$.

In order to find the generalised form of a binary-to-q-nary RM expansion, consider one-variable case first. According to Definition 2.13, one-variable RM expansion for q-nary function $f(x)$ takes the form:

$$f_{q\to q}(\widetilde{x}) = a_0 + a_1\widetilde{x} + \ldots + a_{p-1}\widetilde{x}^{q-1} \quad \text{over } GF(q) \tag{4.12}$$

Constraining variable x and literal $\widetilde{x}$ to the values 0 and 1, we have $\widetilde{x} = \widetilde{x}^2 = \ldots = \widetilde{x}^{q-1}, x \in \{0,1\}, \widetilde{x} \in \{0,1\}$. In other words, $\widetilde{x}^m = \widetilde{x}$ for any nonzero m. Hence (4.12) can be rewritten as (4.13) giving us the form of one-variable binary-to-q-nary RM

expansion.

$$\begin{aligned} f_{2\to q}(\widetilde{x}) &= a_0 + \left(a_1 + \ldots + a_{q-1}\right)\widetilde{x} \\ &= b_0 + b_1\widetilde{x} \quad \text{over GF}(q) \\ b_0 &= a_0 \\ b_1 &= a_1 + \ldots + a_{q-1} \end{aligned} \tag{4.13}$$

Similarly to $p \to p^t$ case, we can use factorisation[1] to reduce the number of terms to 2^n. Since $\widetilde{x_j} = \widetilde{x_j}^2 = \ldots = \widetilde{x_j}^{q-1}$ is true for all literals $\widetilde{x_j} \in \{0,1\}$, $j = 1 \ldots n$; thus each term in (2.1) is transformed as follows.

$$a_i \widetilde{x_1}^{i_1} \ldots \widetilde{x_n}^{i_n} = a_i \widetilde{x_1}^{\psi(i_1)} \ldots \widetilde{x_n}^{\psi(i_n)}$$

$$\psi(i_j) = \begin{cases} 0 & \text{if } i_j = 0 \\ 1 & \text{if } i_j > 0 \end{cases}$$

Finally, the fixed polarity Reed-Muller expansion over GF(q) of the q-nary function $f(x_1, \ldots, x_n)$ takes the form (4.14) if each of the variables $x_j \in \{0,1\}$ and $\widetilde{x}_j \in \{0,1\}$; $\{0,1\} \subset \text{GF}(q)$.

$$f_{2\to q}(\widetilde{x}_1, \ldots, \widetilde{x}_n) = \sum_{i=0}^{2^n-1} b_i \left[\prod_{j=1}^{n} \widetilde{x}_j^{i_j}\right] \quad \text{over GF}(q) \tag{4.14}$$

where i is an integer representation of a binary tuple $\langle i_n \ldots i_1 \rangle_2 = i$, and vector $b = \begin{bmatrix} b_0 & \ldots & b_{2^n-1} \end{bmatrix}^t$ is a coefficient vector, which can be derived from coefficient vector

[1] In general, factorisation may change the polarity, for example, $x + xy = x(1+y) = x\overline{y}$. However, this is not the case in factorisation we use.

a as follows:

$$\begin{aligned} b_i &= \sum_{m=0}^{q^n-1} \omega_{i,m} a_m, i = 0 \dots 2^n - 1 \\ \omega_{i,m} &= \begin{cases} 1 & \text{if there exists such lthat } i_l = \psi(m_l) \\ 0 & \text{otherwise} \end{cases} \\ \langle i \rangle_{10} &= \langle i_n \dots i_1 \rangle_2 \\ \langle m \rangle_{10} &= \langle m_n \dots m_1 \rangle_q \end{aligned}$$

Indeed, in the case of (4.13) for $n = 1$:

$$b_i = \alpha_{i,0} a_0 + \dots + \alpha_{i,q-1} a_{q-1}, i \in \{0, 1\}$$

$$\begin{array}{cccccc} \alpha_{0,0} & = & 1 & \alpha_{1,0} & = & 0 \\ \alpha_{0,1} & = & 0 & \alpha_{1,1} & = & 1 \\ & \dots & & & \dots & \\ \alpha_{0,q-1} & = & 0 & \alpha_{1,q-1} & = & 1 \end{array}$$

Hence, the binary-to-q-nary expansion has a form of a *binary* RM expansion computed over GF(q), and the coefficient vector b can be found from the truth vector using a modified direct method as described below.

Let $d = \begin{bmatrix} d_0 & d_1 \end{bmatrix}^t$ be the truth vector of the binary-to-q-nary RM expansion (4.13); $d_0, d_1 \in GF(q)$. Then, for 0-polarity $\widetilde{x} = x$ and:

$$\begin{array}{ccccc} d_0 & = & f_{2\to q}(0) & = & b_0 \\ d_1 & = & f_{2\to q}(1) & = & b_0 + b_1 \end{array} \quad \text{over } GF(q)$$

By solving this simple set of equations, b_0 and b_1 are found as follows:

$$\begin{array}{ccc} b_0 & = & d_0 \\ b_1 & = & (-1)\, d_0 + d_1 \end{array} \quad \text{over } GF(q)$$

or in matrix form:

$$\begin{aligned} b &= Q_0 d \\ Q_0 &= \begin{bmatrix} 1 & 0 \\ (-1) & 1 \end{bmatrix} \end{aligned}$$

where 0, 1 and (-1) are elements of GF(q); $(-1)+1=0$.

For 1-polarity, $\widetilde{x} = (-1)\,x + 1$, and:

$$\begin{aligned} d_0 &= f_{2\to q}\left((-1)\cdot 0+1\right) = f_{2\to q}(1) = b_0 + b_1 \\ d_1 &= f_{2\to q}\left((-1)\cdot 1+1\right) = f_{2\to q}(0) = b_0 \end{aligned}$$

$$\begin{aligned} b_0 &= d_1 \\ b_1 &= d_0 + (-1)\,d_1 \end{aligned}$$

$$\begin{aligned} b &= Q_1 d \\ Q_1 &= \begin{bmatrix} 0 & 1 \\ 1 & (-1) \end{bmatrix} \end{aligned}$$

Matrices Q_0 and Q_1 can be henceforth used to compute n-variable binary-to-q-nary RM expansions. In matrix form (4.14) can be expressed as follows:

$$f_{2\to q}(\widetilde{x}_1, \ldots, \widetilde{x}_n) = \left(\widetilde{X}_n \otimes \ldots \otimes \widetilde{X}_1\right) \cdot b \quad \text{over GF}(q)$$

where $\widetilde{X}_j = \begin{bmatrix} 1 & \widetilde{x}_j \end{bmatrix}$ is the vector of all possible powers (0 and 1) of the literal $\widetilde{x}_j$ since $x_j \in \{0,1\}$ and $\widetilde{x}_j \in \{0,1\}$; $\{0,1\} \subset \text{GF}(q)$.

From the properties of the Kronecker product, since the whole expression is computed over GF(q) preserving the algebraic properties, coefficient vector b can be found using the following equation.

$$b = \left(Q_{k_n} \otimes Q_{k_{n-1}} \otimes \ldots \otimes Q_{k_1}\right) \cdot d \quad \text{over GF}(q)$$

where the polarity number k refers to binary literals, i.e. $\langle k \rangle_{10} = \langle k_n, k_{n-1}, \ldots, k_1 \rangle_2$.

This research is not aimed at optimising the computational complexity of the algorithm. However, it can be noted that the matrices Q_0 and Q_1 are similar to those of the uniform binary case, so we believe that the optimisation techniques used for the binary radix can be adopted to binary-to-q-nary RM expansions.

The following example illustrates the structure of the circuit implementing binary-to-q-nary RM expansions.

Example 4.5. For GF(3), the element $(-1) = 2$, so the matrices Q_0 and Q_1 are:

$$Q_0 = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$$

For binary-to-ternary expansion the vectors $d_{0,2\to3} = [00220012]^t$, $d_{1,2\to3} = [11001100]^t$ are taken, and the result is as follows ($k = 0$):

$$b_1 = (Q_0 \otimes Q_0 \otimes Q_0) \cdot d_{0,2\to3} \quad \text{over GF(3),},$$

$$b_1 = \left(\begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \right) \cdot d_{0,2\to3}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 2 & 1 & 0 & 0 \\ 2 & 0 & 2 & 0 & 1 & 0 & 1 & 0 \\ 1 & 2 & 1 & 2 & 2 & 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 2 \\ 2 \\ 0 \\ 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 2 \\ 1 \end{bmatrix},$$

$$b_2 = (Q_0 \otimes Q_0 \otimes Q_0) \cdot d_{1,2\to3} = [10200000]^t \quad \text{over GF(3),}$$

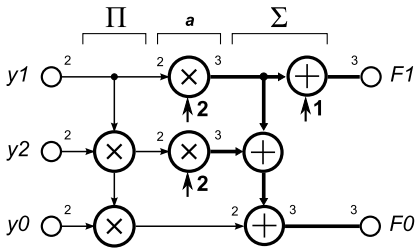


Figure 4.6: Component-level schematic for binary-to-ternary RM expansion.

and the functions take the form:

$$\begin{aligned} F_{0,2\to3}(y_0, y_1, y_2) &= 2y_1 + 2y_1y_2 + y_0y_1y_2 \\ F_{1,2\to3}(y_0, y_1, y_2) &= 1 + 2y_1 \end{aligned}$$

The schematic is shown in Figure 4.6.

4.6 Summary

A radix model is a schema of applying radices at the level of mathematical representation, which is chosen before the synthesis stage, and the circuit is produced with respect to the radix model from the very beginning. For the sum of product, two-level radix models imply application of one radix to the operations of multiplication and another radix to the operations of addition. Such simplification narrows down the radix search space and enables cross-radix computation.

A number of two-level radix models has been developed and described. Due to the homomorphism of certain fields, it was possible to extend the Green's direct method to compute mixed radix Reed-Muller expansions in the presented radix models. The possibility to apply faster methods of computation has not been explored so far, as the estimation of the mixed radix efficiency has been put at higher priority. The described method has been effective enough to synthesise the benchmarks presented in Chapter 5.

This chapter is mostly based on the journal paper accepted for publication [53].

Mixed radix domain model has been published for $\{2, 4\} \rightarrow 4$ case in [58].

Chapter 5

Use Case and Experimental Results

The presented research is focused on the security applications. However, the MVL synthesis approach based on mixed radix Reed-Muller expansions over Galois fields can be applied to any area that requires encoding of signals and multi-valued logic.

This chapter presents the application of the theory described in Chapter 4. In order to proceed with the proposed design flow, the designer must follow the steps described below.

1. Define encoding. The encoding is the foundation of the approach. If there is no need to encode the signals, the standard EDA design flow is recommended instead. In our case, 1-of-n encoding has been chosen, as discussed in Chapter 2.

2. Build the component library of GF operations with respect to the specified encoding and other application requirements. Include addition $(x + y)$, multiplication $(x \cdot y)$, and operations with constants $(x + c)$, $(x \cdot c)$ over selected Galois fields. Estimate the physical properties of the components using per-component SPICE simulation or another acceptable technique. Section 5.1 gives the example implementations for the security applied 1-of-n encoded libraries.

3. From the physical properties of the designed component libraries, choose the appropriate radix models.

4. Map function specifications into required radices. Ideally, this could be done by a synthesis tool – we consider it as one of the subjects for future work.

5. Use the provided synthesis tool (presented in Section 5.3) to produce the encoded data path.

6. Integrate data path modules with the control path as proposed in Chapter 3 and continue with the regular EDA procedures such as place and route.

The Section 5.4 discusses the security applied benchmark results with respect to the component level and the technology level parameters.

5.1 Component implementations

Chapters 2 and 4 have presented the synthesis at the level of technology-independent components. The components thereafter are substituted by their physical implementation using predesigned libraries of GF arithmetic components. These libraries constitute a connection between the abstract mathematical representation and the technology level. Applying the theory of RM expansions to secure devices we compare two approaches to 1-of-n encoded logic: implementation using runtime library (RTL) cells and custom design dynamic logic cells. This chapter describes these implementations with respect to the features of power-balanced logic.

5.1.1 Implementation using RTL

Generic approaches for 1-of-n codes over Galois fields were patented in [73]. The general guidelines for power balancing in CMOS circuits is given in Chapter 2. Galois field arithmetic components implement the truth tables shown in Figure 2.10 with applied encoding according to Table 2.1.

The numbering of wires encoding a signal is respective, i.e. for 1-of-n encoded n-ary signal x, wire $x_i = 1$ while in the data phase means that $x = i$; $i = 1 \dots n$. The corresponding equations and the list of figures displaying the corresponding RTL

Table 5.1: GF component implementations

radix, encoding	operation	equation	shown in
GF(2), dual-rail	$q = x + y$	$q_0 = x_0y_0 + x_1y_1$ $q_1 = x_0y_1 + x_1y_0$	Figure 5.1
	$q = xy$, relaxed balancing	$q_0 = x_0 + y_0$ $q_1 = x_1y_1$	Figure 2.6(a)
	$q = xy$, fully balanced	$q_0 = x_0y_0 + x_0y_1 + x_1y_0$ $q_1 = x_1y_1 + 0 + 0$	Figure 2.6(b)
GF(3), 1-of-3	$q = x + y$	$q_0 = x_0y_0 + x_1y_2 + x_2y_1$ $q_1 = x_0y_1 + x_1y_0 + x_2y_2$ $q_2 = x_0y_2 + x_1y_1 + x_2y_0$	–
	$q = xy$, relaxed balancing	$q_0 = x_0 + y_0$ $q_1 = x_1y_1 + x_2y_2$ $q_2 = x_1y_2 + x_2y_1$	–
	$q = xy$, fully balanced	$q_0 = x_0y_0 + x_0y_1 + x_0y_2 + x_1y_0 + x_2y_0$ $q_1 = x_1y_1 + x_2y_2 + 0 + 0 + 0$ $q_1 = x_1y_2 + x_2y_1 + 0 + 0 + 0$	–
GF(4), 1-of-4	$q = x + y$	$q_0 = x_0y_0 + x_1y_1 + x_2y_2 + x_3y_3$ $q_1 = x_0y_1 + x_1y_0 + x_2y_3 + x_3y_2$ $q_2 = x_0y_2 + x_1y_3 + x_2y_0 + x_3y_1$ $q_3 = x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0$	Figure 5.2
	$q = xy$, relaxed balancing	$q_0 = x_0 + y_0$ $q_1 = x_1y_1 + x_2y_3 + x_3y_2$ $q_2 = x_1y_2 + x_2y_1 + x_3y_3$ $q_3 = x_1y_3 + x_2y_2 + x_3y_1$	Figure 5.3
	$q = xy$, fully balanced	$q_0 = x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_0 + x_2y_0 + x_3y_0$ $q_1 = x_1y_1 + x_2y_3 + x_3y_2 + 0 + \ldots + 0$ $q_2 = x_1y_2 + x_2y_1 + x_3y_3 + 0 + \ldots + 0$ $q_3 = x_1y_3 + x_2y_2 + x_3y_1 + 0 + \ldots + 0$	Figure 5.4

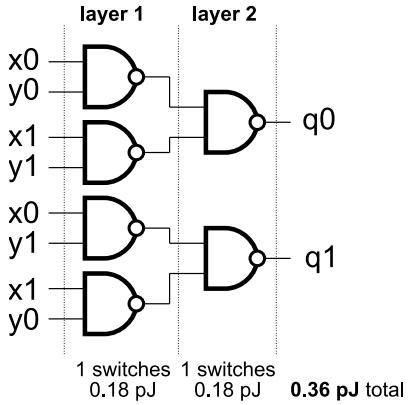


Figure 5.1: Dual-rail encoded GF(2) addition

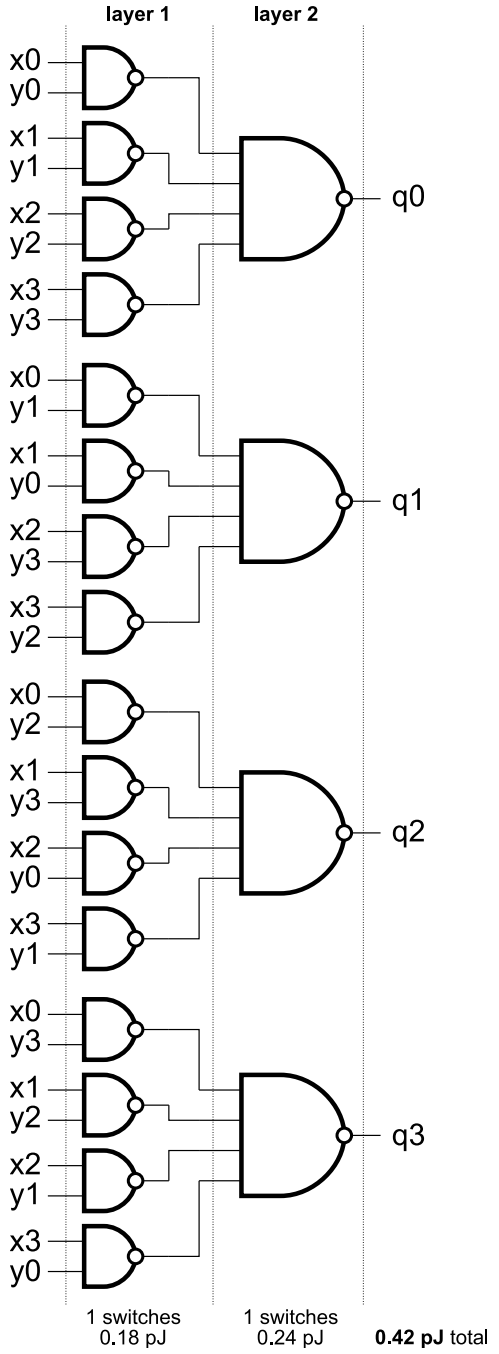


Figure 5.2: 1-of-4 encoded GF(4) addition

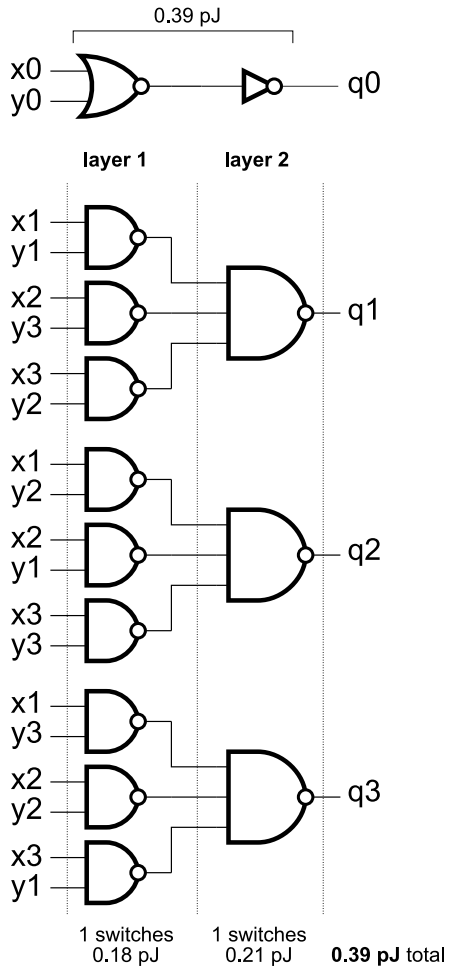


Figure 5.3: 1-of-4 encoded GF(4) multiplication, relaxed balancing

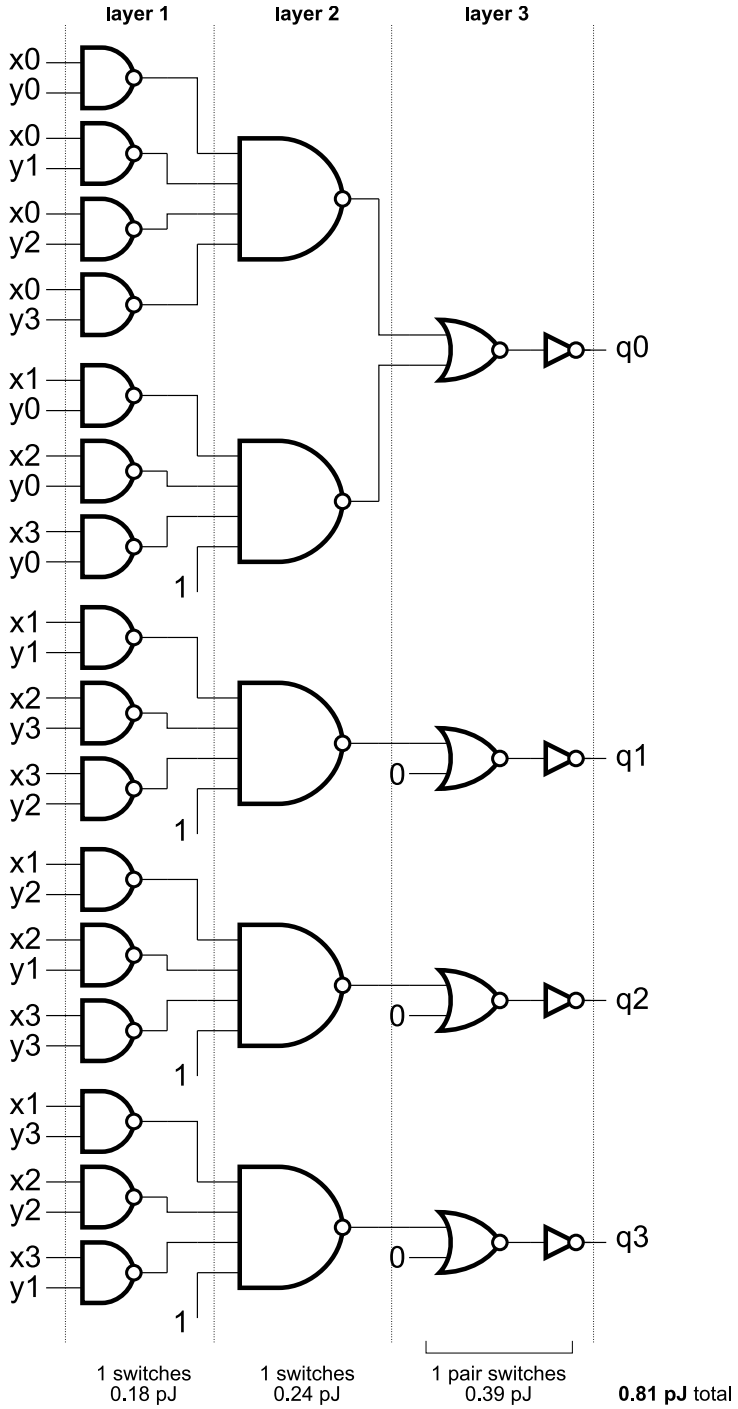


Figure 5.4: 1-of-4 encoded GF(4) multiplication, fully balanced

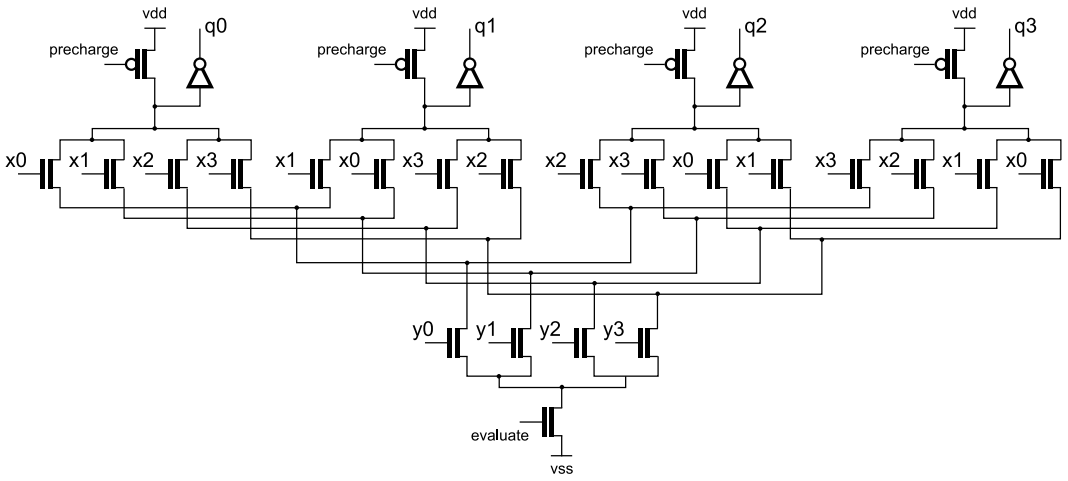


Figure 5.5: Dynamic logic implementation of 1-of-4 encoded GF(4) addition

implementations are given in Table 5.1. The components have eager evaluation of data and early propagation of spacers, i.e. the output signal is evaluated only when the data arrives at both inputs and is reset to NULL right after any of the inputs is dropped to NULL. Higher radix operations have more complex implementations, especially if the balancing feature is fully applied.

5.1.2 Dynamic logic implementation

In order to obtain better characteristics of balanced components, we use custom design cells that optimise area and switching behaviour at the transistor level. The 1-of-n library described in [16] builds GF arithmetic components using dynamic logic [78].

An example of a dynamic logic based one-hot encoded GF component is shown in Figure 5.5. In CMOS the use of higher radix encoding is well known; evaluations are performed in N-channel logic. Dynamic logic requires two phases. The first phase is called the precharge phase and the second phase the evaluation phase. There are three important benefits to N-channel only evaluation gates relative to traditional static gates [33]. The first is the elimination of P-channel devices on input signals, which reduces the input load significantly. The second is the elimination of the need to build the complementary function in P-channel devices, which means that the more efficient

faster N-channel gates are possible. The third advantage of N-channel only evaluation is the ability to share portions of the evaluate "stack" among multiple outputs, which is not possible with static CMOS gates because it is not possible to obtain each output's function and complement from shared devices in both the P and N-channel stacks.

5.1.3 Estimating component physical parameters

According to the motivation behind the research, the secure design flow necessitates estimation of physical characteristics in order to compare the efficiency of the synthesised expansions. Precise physical simulation is available only after the placement and routing has been done, which is a complex task. At the logic synthesis stage a tool should use statistical evaluation based on numeric values from the library specification.

In our RTL-based implementations we used the Faraday 90nm library (FSD0A_A for UMC's 90nm 1P9M Logic/Mixed Mode Low-K SP process). Energy and area estimations of the proposed components are shown in Table 5.2. In order to show the difference between technologies, we have included the out-dated AMS C35 0.35μm library results as well. For RTL implementations, the total switching energy is computed as the sum of the switching energies for each layer of a component circuit. The total area of a component is a sum of area values for all gates in its circuit. Higher radix components grow too large and difficult to implement, so the possible error in estimating their parameters can be significantly increased. Therefore we limit ourselves to radices 2, 3 and 4.

The dynamic logic library uses a 90nm technology, and its switching energy estimates have been normalised in order to enable cross-technology comparison with 90nm RTL components. GF(2) addition, identical at the transistor level to an RTL implementation, has been simulated in SPICE to find the scaling factor. The component's power consumption has been measured and multiplied by this factor. Although the estimation method is fairly rough, it satisfies the required level of approximation. Area is measured by transistor count using relative measures.

Table 5.2: Physical characteristics of GF components

radix, encoding	operation	physical characteristics					
		RTL: AMS 0.35μm		RTL: Faraday 90nm		Dynamic logic 90nm	
		energy, pJ	area, μm^2	energy, 10^{-3} pJ	area, cell units	energy, 10^{-3} pJ	area, cell units
GF(2), dual-rail	+	0.36	330	3.348	24	2.450	13
	× relaxed balancing	0.39	182	2.783	14	–	–
	× fully balanced	0.39	366	4.110	28	2.430	13
GF(3), 1-of-3	+	–	–	4.110	54	2.640	18
	× relaxed balancing	–	–	3.348	31	–	–
	× fully balanced	–	–	6.893	81	2.410	24
GF(4), 1-of-4	+	0.42	1244	7.419	108	2.520	27
	× relaxed balancing	0.39	805	4.110	61	–	–
	× fully balanced	0.81	1699	10.202	147	2.950	39

The parameters of 0.35μm-implemented components cannot be compared to other libraries as absolute values, since they are obtained using different estimations rules. However, it is still possible to observe how different radices impact on the component and circuit parameters for this technology.

As can be seen from Table 5.2, RTL operations over GF(4) show a considerable area overhead compared to their GF(2) counterparts. A possible explanation is the fact that binary operations use the same radix as their gate level implementations, so the radix mapping problem is encountered at the sub-component level. Dynamic logic however has reduced overheads in higher radices, especially for switching energy, as it is less dependent on the binary computation according to the features listed in Section 5.1.2. Interestingly, the AMS library has acceptably cheap energy costs for quaternary components, while in Faraday both area and power go up with the radix.

Another interesting point concerning 1-of-n encoded component implementations is so-called "wire crossing" operations. Certain arithmetic functions in these encodings degenerate into trivial re-assignment of signals. For example, dual-rail inversion $(x+1)$ has the following structure: $q_0 = x_1$, $q_1 = x_0$. The truth table for "wire crossing" operations over GF(4) is shown in Figure 5.6. In the one-hot ternary case such trivial operations are: $(x+1)$, $(x+2)$ and $2x$. Consequently, these pseudo-components have

x	0	1	A	B
$x+1$	1	0	B	A
$x+A$	A	B	0	1
$x+B$	B	A	1	0
Ax	0	A	B	1
Bx	0	B	1	A
x^2	0	1	B	A

Figure 5.6: "Wire crossing" operations in 1-of-4: their implementations do not involve any logic

Table 5.3: Estimated physical characteristics of the examples using Faraday 90nm library (relaxed balancing)

radix	circuit	switching wires	sw. en., 10^{-3}pJ	area, c. units
$2 \to 2$	Figure 2.11 on page 38	6	18.393	114
$2 \to 3$	Figure 4.6 on page 96	4	13.786	136
$2 \to 4$	Figure 4.4 on page 85	3	12.985	136
$\{2,4\} \to 4, r = 1$	Figure 4.5(a) on page on page 90	5	23.187	286
$\{2,4\} \to 4, r = 2$	Figure 4.5(b) on page on page 90	5	28.638	347
$4 \to 4$	Figure 2.12 on page 40	8	52.734	942
$4 \to 2$	Figure 4.3 on page 78	8	49.120	587

no area or switching costs, and this also should be considered when estimating physical circuit parameters.

From the discussion presented in this subsection one can conclude that simple efficiency analysis, such as the number of terms or the number of operations, is not sufficient for the correct search for the optimal RM expansion. All components have specific characteristics that depend on the choice of the library, encoding and power balancing strength. The next subsection presents and compares benchmark results for different radix combinations.

Example 5.1. Assuming that Examples 2.1—4.5 are required to be secure (power balanced), the components have been mapped into RTL implementations from Table 5.2. Table 5.3 gives the estimates for the physical parameters of the circuits. Although the mixed radix examples show fewer components and reduced switching activity, the area for binary is smaller, as expected from the background discussion.

The next section presents and compares benchmark results for real security algorithms, and also gives a detailed discussion on how we estimate the circuit parameters.

5.2 Mapping specifications from one radix to another

For a combinational logic circuit defined as a p-nary truth table, the goal is to build the truth table in radix q with the same functionality. Formally this can be defined as follows. For a given p-nary mapping $F_{p\to p} : Y \to E$ the task is to build a q-nary mapping $F_{q\to q} : X \to D$, such that having defined radix interpretations $\rho_1 : Y \to X$ and $\rho_2 : E \to D$, we have

$$F_{q\to q}(\rho_1(y)) = \rho_2(F_{p\to p}(y)) \tag{5.1}$$

for each $y \in Y$. It should be noted that this is not a general isomorphism, as ρ_1 and ρ_2 are not necessarily isomorphic mappings.

The radix interpretation $\rho_1 : Y \to X$ can be defined in a trivial way:

$$y_{n_1-1}p^{n_1-1} + \ldots + y_1 p + y_0 = x_{n_2-1}q^{n_2-1} + \ldots + x_1 q + x_0 \tag{5.2}$$

where $y_{n_1-1}\ldots y_0$ are p-nary digits of $y \in Y$ and $x_{n_2-1}\ldots x_0$ are q-nary digits of $x \in X$. For example, for $p = 2$, $q = 3$ and $y = \langle 1011\rangle_2$, we have $2^3 + 2^1 + 2^0 = 3^2 + 2\cdot 3^0$, hence $x = \langle 102\rangle_3$. Similarly for $\rho_2 : E \to D$.

The cardinality of the domain Y of the function $F_{p\to p}$ is $|Y| = p^{n_1}$, where n_1 is the number of p-nary inputs, and the cardinality of its co-domain E is $|E| = p^{m_1}$, where m_1 is the number of p-nary outputs. For the function $F_{q\to q}$, $|X| = q^{n_2}$ and $|D| = q^{m_2}$ respectively. The number of inputs n_2 and outputs m_2 must be chosen to guarantee that the domain and co-domain of $F_{q\to q}$ "cover" the domain and co-domain of the original function, in other words, $|X| \geqslant |Y|$ and $|D| \geqslant |E|$. Consequently, $q^{n_2} \geqslant p^{n_1}$ and $q^{m_2} \geqslant p^{m_1}$.

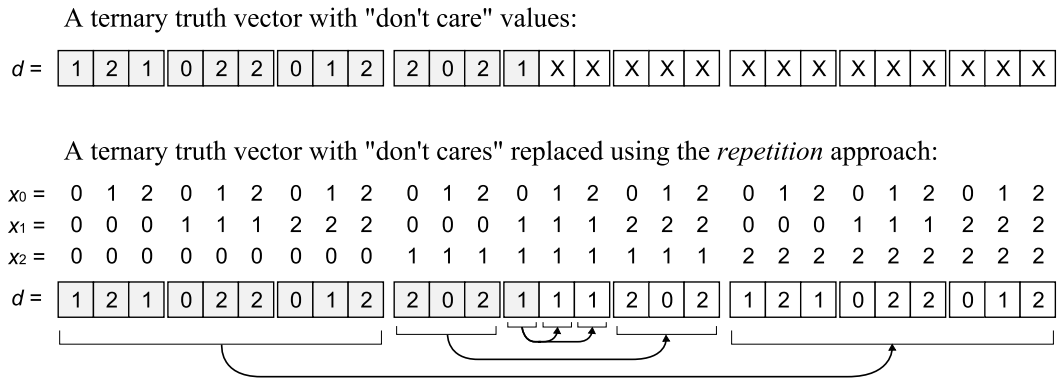


Figure 5.7: An example of using the *repetition* heuristics to handle "don't care" values.

Dealing with "don't care" values The cardinality of the domain determines the length of the truth vector. The truth table of $F_{q\to q}$ can be longer than the truth table of $F_{p\to p}$, so it may contain "don't care" (DC) values[1]. Expansions from (2.1) and (4.14) present one digit q-nary outputs, thus working with truth vectors instead of truth tables. Here a truth vector is a single column (digit) of the truth table. Unfortunately, there is no straight-forward solution to deal with DC values in multi-valued RM expansions, so each DC value requires an exhaustive search in order to reduce the logic overheads. Hence the computation time is increased q^{μ} times, where μ is the number of DC values in the truth vector, which can be significantly large. Heuristics should be used instead.

Simply replacing all the DC values with zeroes or any other arbitrary constant generates rather poor results. For example, in the case of the binary truth vector [10XX], replacement with [1000] results in $f(x_1, x_2) = \overline{x}_1\overline{x}_2$, while the truth vector [1010] would give the simple solution $f(x_1, x_2) = \overline{x}_1$. The DC values can be filled with repeating power of q sized portions of the truth vector, thus reducing the dependency on certain variables. Figure 5.7 shows an example how this can be done. The target truth vector has length 27, so the function has 3 ternary input variables. The first 9 values in the truth vector correspond to $x_2 = 0$. Copying these values to the last 9 positions of the truth vector ($x_2 = 2$) make the whole function less dependant on x_2. The next

[1]In this section we do not discuss input "don't cares" as they are the feature of a function specification and do not appear from radix mapping.

3 values correspond to $x_1 = 0, x_2 = 1$ and copied to the positions corresponding to $x_1 = 2, x_2 = 1$, hence the result becomes less dependant on x_1 while $x_2 = 1$. Finally the middle section, $x_0 = 0, x_1 = 1, x_2 = 1$, is copied to the next two positions, so the function is not dependant on x_0 while $x_1 = x_2 = 1$.

Most functions are traditionally defined in binary radix $p = 2$. Binary-to-q-nary function $F_{2\to q} : X \to D$ specifies that X is binary, and since Y is also binary, $\rho_1 : Y \to X$ becomes an equivalence: $X \equiv Y$. In other words, binary-to-q-nary function $F_{2\to q}$ uses the same domain as binary function $F_{2\to 2}$. Consequently, $|X| = |Y|$, $n_2 = n_1$, the length of their truth tables are equal, and DC values and all related problems are never encountered, which can be considered as an advantage of binary-to-q-nary functions.

5.3 Synthesis tool and optimisations

The ideas presented in Chapter 4, as well as the synthesis of uniform radix RM expansions, have been implemented in a tool [1] and used to synthesise a number of benchmarks, presented in Section 5.4. Section A.5 also provides an example of using the tool within the design flow. The user manual is given in Appendix A.

Figure 5.8 shows a block diagram showing the processing of one Reed-Muller expansion. The flow consists of the distinct steps described below.

Computing RM expansion

For the given polarity number $k = 0 \dots q^n - 1$ the tool computes the RM expansion in the specified radix. The output of the tool is a coefficient vector, i.e. the RM expansion is given in a mathematical form.

Mapping into GF arithmetic components

The mathematical expression is decomposed into the operations of multiplication $(x \cdot y)$, addition $(x + y)$, multiplication by a constant (cx), and constant addition $(x + c)$ over

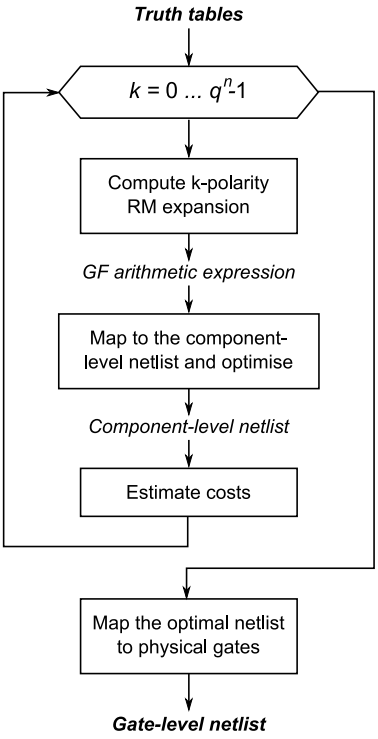


Figure 5.8: Reed-Muller based synthesis flow

GF(2) or GF(q), where x, y are 2-valued or q-valued variables; c is a constant value. The output of this process is a component-level netlist.

Applying component-level optimisations

Logic optimisation, such as minimisation of terms, has to be applied at the component-level rather than the gate-level in order to guarantee that it does not affect the representation of multi-valued signals. Optimisation techniques are the subject of ongoing research. However, a number of basic algorithms have been added to the tool.

The optimisation applied after the decomposition process is a minimisation of the RM expansion. The minimisation problem may also refer to factorisation. However, factorisation is not applicable to the described mixed radix circuits, since it changes the order of additions and multiplications overriding operation radices. We used a first-order minimisation algorithm which extracts repeating subterms and treats them as

temporary variables. A certain research effort has been previously made in the area of uniform radix Reed-Muller expansions [75, 79, 77, 34], transfer of these techniques to the mixed radix approach is one of the suggestions for future work.

Additional type of optimisation reduces the radix of q-nary signals that are constrained to less than q values. For example, according to the properties of GF(q), $x^{q-1} = 1$ for any non-zero x thus constraining the result of this operation to the binary range. Consequently all $q-1$ power forms of the variables in uniform q-nary RM expansions can use multiplications over GF(2) instead of GF(q). Since GF(q) multiplication of binary arguments also produce binary results, GF(2) propagation through the circuit can be traced. This optimisation approach does not alter the circuit structure: it attempts to remove paths that never switch. Hence it reduces the area whilst the energy consumption is not affected.

Estimating circuit costs

Statistical information on arithmetic components can be collected after the optimisation is done and then used to approximately estimate the actual physical characteristics of the synthesised circuit. The estimated cost is used to search for the most efficient implementation through q^n polarity forms.

Typically the expansion with the least number of non-zero terms is chosen as the best one [28, 59]. In this case, the actual number of additions and multiplications as well as their power and area costs is not considered. The exact number of operations is known only after the decomposition. For large circuits the execution time can be infeasible if we decompose the expansions for all polarities. Therefore the decomposition should be performed for a smaller number of the best expansion candidates selected using a simple suboptimal criterion, e.g. the number of non-zero terms.

After the best solution is found, the components are mapped directly to gates using the predefined component library.

Table 5.4: Mixed domain results for different radix numbers; r* is the best radix number found

	$2 \rightarrow 4$		$\{2,4\} \rightarrow 4$		$4 \rightarrow 4$	
circuit	r	operations	r*	operations	r	operations
aes	15	1427	**14**	1391	0	1405
misty7	15	151	15	151	0	338
misty9	31	165	31	165	0	482
kasumi7	15	144	15	144	0	359
kasumi9	31	161	31	161	0	454
des1	7	151	7	151	0	153
des2	7	123	7	123	0	143
des3	7	131	7	131	0	149
des4	7	152	**1**	149	0	161
des5	7	137	**6**	132	0	137
des6	7	136	7	136	0	141
des7	7	131	**5**	125	0	152
des8	7	122	7	122	0	147

5.4 Synthesis results

DES, AES [9], Serpent [12], Kasumi [8] and Misty [44] S-boxes have been chosen as benchmarks because they are commonly used cryptographic algorithms. Typically they are specified in the form of a truth table, which is convenient for the RM approach. These circuits have been synthesised in binary, ternary, quaternary radices and also their two-level combinations with binary using the approach presented here.

It is important that binary circuits have been synthesised using the same flow as higher radix examples, since we intend to compare the effect of radices rather than the synthesis algorithms. According to the discussion in Chapter 1 and the peculiarity of the application area, we intentionally do not compare the results with the conventional tools, such as Synopsys and Cadence, which simply cannot be used in this case. We do not think the mixed radix approach can replace the standard binary flow in general purpose logic synthesis. The goal of the research, as well as its main potential, is in reinforcing the specialised design flow where the specific signal representation is a dominant prerequisite.

Before giving the discussion on the main results, let's consider some intermediate

results produces while the tool was at the initial state of its development. Table 5.4 shows the synthesis results for the mixed radix domain radix model $\{2,4\} \rightarrow 4$. The radix number r* has been obtained by searching through all possible radix numbers including $2 \rightarrow 4$ and $4 \rightarrow 4$ cases, thus it corresponds to the best solution by definition. At the component level, the number of operations for the majority of the benchmarks is optimal for $2 \rightarrow 4$ radix model, and only few benchmarks got different radix number. The logic minimisation algorithm in that version of the tool was constrained by the mixed radix domain. However, keeping all input variables in the same radix, enabled further significant improvement of the algorithms. Since the mixed radix domain did not give considerable benefit, the support of this model has been removed from the tool.

The following benchmark runs has been made for uniform binary, ternary and quaternary radices ($2 \rightarrow 2, 3 \rightarrow 3, 4 \rightarrow 4$) and the radix extension models ($2 \rightarrow 3, 2 \rightarrow 4$). We have considered radix reduction model as well: the tool is able to compute $4 \rightarrow 2$ truth vector and map it to the library components, but the current version does not give a satisfactory optimisation of such a circuit, so the comparison with other radix models would be unfair. A proper minimisation of radix reduction circuits is a subject of futher improvement of the tool. Therefore, no radix reduction model has been included in the benchmarks results.

For the selected radix models, during the synthesis process the tool searches through all possible polarity forms for each benchmark. Each candidate is decomposed into a netlist and optimised, before estimating its cost. In spite of the high complexity, such a search is feasible even for relatively large benchmarks: for AES, 2^8 polarity forms, each computing 2^8 long coefficient vectors for every one of 8 (binary) outputs have been synthesised less than a minute on a 1.60 GHz Intel i7-720Q laptop (the number of cores is not relevant as the tool is currently does not use parallel computation). Runtimes for all benchmarks as well as their sizes are given in Tables 5.5–5.7. Note that the tool uses the computation based on the direct method, so the exponential computational complexity significantly limits the size of the benchmarks: the largest

synthesised circuits are Kasumi 9 and Misty 9 S-boxes with 9 binary inputs and 9 binary outputs. Optimisation of computation time is out of the scope of the thesis and considered as a part of future work.

At the technology mapping stage, GF components have been substituted with their RTL implementations and dynamic logic implementations presented in Section 5.1. Tables 5.5–5.7 show a number of selected characteristics for the component level, namely the number of switching wires (sw. wires), number of operations per radix, critical path length. The switching is given for the full period, i.e. each component switches twice: from spacer to data and from data back to spacer. Technology mapped results are shown in Tables 5.8–5.10 giving the switching energy of gates (sw. en.) and total area. As has been mentioned, precise evaluation of physical characteristics requires placement and routing to be done. In our case an approximate statistical estimation (described below) based on the library specifications has been used instead.

For the balanced benchmarks (dynamic logic and relaxed balancing in Faraday 90nm and AMS 0.35 μm^2), all signals are encoded using corresponding 1-of-n code. Single-rail results are given as a reference point and are not discussed futher in this section.

Since the power balanced implementation has the property of data-independent energy consumption, the *total switching energy of gates* in the circuit per period can be approximated to a constant and found as a sum of switching energies of all components according to their library specification given in Table 5.2. For single-rail circuits the switching energy is also given as a total sum, thus providing the maximum switching energy in the circuit. During the normal operation in the single-rail circuit only 10% of gates switch , and they switch at a half speed comparing to 1-of-n codes.

Per-radix component count considers all components including wire-crossing operations, therefore their sum exceeds the number of switching wires. It is interesting that binary operations can be found in uniform radix MVL circuits. This is the result of q-nary operation x^{q-1}, hence according to Section 5.3, such signals have been reduced in radix in order to optimise the circuit area.

The given *area* parameter is a sum of the component areas. It does not include the area occupied by the wires, as this is unknown at this stage. The circuit delay is estimated at the component level as the length of the critical path, and given as a number of components.

The presented results do not show a distinctively optimal radix solution, however certain general trends can be observed and analysed. The efficiency of the synthesised circuits depends on the properties of the implemented functions. Unfortunately, there is no straight-forward solution to predict a function's behaviour in the target radix from its truth table. RM design flow however is not bound to any particular radix, so it can be used to investigate this after the synthesis is done. The presented tool provides a range of characteristics giving a full picture how the function is related to certain radices or their combinations.

The original specifications for the S-boxes are given in a binary radix. In order to be synthesised in higher radices, the truth tables had to be converted. For an n_1-variable q-nary function, q^{n_1} is the cardinality of its domain (and the length of its truth vector). While converting to radix p, we need to ensure that the whole domain is covered by a new specification, i.e. $p^{n_2} \geqslant q^{n_1}$, where n_2 is the number of arguments of the new function. In the case of $p^{n_2} > q^{n_1}$, the final truth vector is longer than the original one and may contain "don't care" values. The problem of "don't care" values is not covered by this thesis, but in general, longer truth vectors normally result in larger circuits.

For the component level benchmarks shown in Tables 5.5–5.7, one can observe good switching activity of wires for quaternary and binary-to-quaternary circuits. In terms of specification, the benchmarks are useful for quaternary, as they have 8, 6 and 4 binary inputs and the same number of outputs; $2^8 = 4^4$, $2^6 = 4^3$ and $2^4 = 4^2$, so the radix mapping does not induce "don't cares" and any associated overheads. The reason for the component-level overheads in binary-to-ternary and ternary circuits is inefficient domain (only for ternary) and co-domain mapping: 2^8, 2^6 and 2^4 do not have matching 3^m substitutes.

The fact that binary-to-quaternary results are better than purely quaternary ones

Table 5.5: Component level synthesis results

circuit	radix	truth vectors, number and length	runtime, sec	component level generic parameters			
				sw. wires	operations		crit. path, comp-s
					total	by radix	
des1	$2 \rightarrow 2$	4×2^6	0.6	330	171	2:171	10
	$2 \rightarrow 3$	3×2^6	0.8	362	232	2:87; 3:145	10
	$3 \rightarrow 3$	3×3^4	1.2	396	247	2:16; 3:231	11
	$2 \rightarrow 4$	2×2^6	0.5	246	173	2:60; 4:113	9
	$4 \rightarrow 4$	2×4^3	0.5	**244**	173	2:5; 4:168	9
des2	$2 \rightarrow 2$	4×2^6	0.5	258	135	2:135	8
	$2 \rightarrow 3$	3×2^6	0.8	350	224	2:64; 3:160	10
	$3 \rightarrow 3$	3×3^4	1.1	372	227	2:15; 3:212	11
	$2 \rightarrow 4$	2×2^6	0.5	**224**	160	2:59; 4:101	9
	$4 \rightarrow 4$	2×4^3	0.5	252	185	2:5; 4:180	9
des3	$2 \rightarrow 2$	4×2^6	0.5	252	133	2:133	8
	$2 \rightarrow 3$	3×2^6	0.8	376	233	2:118; 3:115	10
	$3 \rightarrow 3$	3×3^4	1.1	378	237	2:15; 3:222	11
	$2 \rightarrow 4$	2×2^6	0.5	**246**	172	2:70; 4:102	10
	$4 \rightarrow 4$	2×4^3	0.5	**246**	177	2:4; 4:173	9
des4	$2 \rightarrow 2$	4×2^6	0.6	340	173	2:173	9
	$2 \rightarrow 3$	3×2^6	0.8	358	224	2:72; 3:152	10
	$3 \rightarrow 3$	3×3^4	1.1	386	243	2:16; 3:227	11
	$2 \rightarrow 4$	2×2^6	0.5	282	203	2:60; 4:143	10
	$4 \rightarrow 4$	2×4^3	0.5	**280**	198	2:7; 4:191	9
des5	$2 \rightarrow 2$	4×2^6	0.5	320	167	2:167	8
	$2 \rightarrow 3$	3×2^6	0.8	358	226	2:89; 3:137	10
	$3 \rightarrow 3$	3×3^4	1.1	378	230	2:16; 3:214	11
	$2 \rightarrow 4$	2×2^6	0.5	**242**	178	2:56; 4:122	10
	$4 \rightarrow 4$	2×4^3	0.5	260	189	2:5; 4:184	9
des6	$2 \rightarrow 2$	4×2^6	0.5	288	149	2:149	8
	$2 \rightarrow 3$	3×2^6	0.8	322	199	2:73; 3:126	10
	$3 \rightarrow 3$	3×3^4	1.1	366	226	2:15; 3:211	11
	$2 \rightarrow 4$	2×2^6	0.5	**254**	173	2:74; 4:99	10
	$4 \rightarrow 4$	2×4^3	0.5	260	190	2:5; 4:185	9
des7	$2 \rightarrow 2$	4×2^6	0.5	270	139	2:139	9
	$2 \rightarrow 3$	3×2^6	0.8	340	219	2:77; 3:142	10
	$3 \rightarrow 3$	3×3^4	1.1	366	223	2:15; 3:208	11
	$2 \rightarrow 4$	2×2^6	0.5	**230**	171	2:48; 4:123	9
	$4 \rightarrow 4$	2×4^3	0.5	254	181	2:5; 4:176	9
des8	$2 \rightarrow 2$	4×2^6	0.5	254	132	2:132	8
	$2 \rightarrow 3$	3×2^6	0.8	356	225	2:85; 3:140	10
	$3 \rightarrow 3$	3×3^4	1.1	346	216	2:16; 3:200	11
	$2 \rightarrow 4$	2×2^6	0.5	**202**	143	2:43; 4:100	10
	$4 \rightarrow 4$	2×4^3	0.5	232	168	2:5; 4:163	9

Table 5.6: Component level synthesis results (continued)

circuit	radix	truth vectors, number and length	runtime, sec	component level generic parameters			
				sw. wires	operations		crit. path, comp-s
					total	by radix	
serp1	$2 \rightarrow 2$	4×2^4	0.1	56	32	2:32	5
	$2 \rightarrow 3$	3×2^4	0.1	70	57	2:18; 3:39	6
	$3 \rightarrow 3$	3×3^3	0.2	78	53	2:7; 3:46	8
	$2 \rightarrow 4$	2×2^4	0.1	**52**	39	2:14; 4:25	6
	$4 \rightarrow 4$	2×4^2	0.1	58	44	2:2; 4:42	6
serp2	$2 \rightarrow 2$	4×2^4	0.1	66	37	2:37	6
	$2 \rightarrow 3$	3×2^4	0.1	78	53	2:22; 3:31	6
	$3 \rightarrow 3$	3×3^3	0.2	84	54	2:8; 3:46	7
	$2 \rightarrow 4$	2×2^4	0.1	**52**	43	2:12; 4:31	6
	$4 \rightarrow 4$	2×4^2	0.1	60	49	2:3; 4:46	6
serp3	$2 \rightarrow 2$	4×2^4	0.1	60	35	2:35	5
	$2 \rightarrow 3$	3×2^4	0.1	76	49	2:19; 3:30	6
	$3 \rightarrow 3$	3×3^3	0.2	68	44	2:8; 3:36	7
	$2 \rightarrow 4$	2×2^4	0.1	**50**	42	2:12; 4:30	6
	$4 \rightarrow 4$	2×4^2	0.1	**50**	41	2:2; 4:39	6
serp4	$2 \rightarrow 2$	4×2^4	0.1	62	35	2:35	5
	$2 \rightarrow 3$	3×2^4	0.1	62	45	2:15; 3:30	6
	$3 \rightarrow 3$	3×3^3	0.2	86	58	2:7; 3:51	7
	$2 \rightarrow 4$	2×2^4	0.1	52	42	2:11; 4:31	6
	$4 \rightarrow 4$	2×4^2	0.1	**48**	41	2:2; 4:39	6
serp5	$2 \rightarrow 2$	4×2^4	0.1	66	39	2:39	6
	$2 \rightarrow 3$	3×2^4	0.1	64	41	2:25; 3:16	6
	$3 \rightarrow 3$	3×3^3	0.2	86	52	2:6; 3:46	8
	$2 \rightarrow 4$	2×2^4	0.1	52	40	2:11; 4:29	6
	$4 \rightarrow 4$	2×4^2	0.1	**44**	33	2:2; 4:31	6
serp6	$2 \rightarrow 2$	4×2^4	0.1	66	39	2:39	5
	$2 \rightarrow 3$	3×2^4	0.1	80	48	2:29; 3:19	6
	$3 \rightarrow 3$	3×3^3	0.2	76	53	2:7; 3:46	8
	$2 \rightarrow 4$	2×2^4	0.1	**52**	41	2:11; 4:30	6
	$4 \rightarrow 4$	2×4^2	0.1	**52**	44	2:4; 4:40	5
serp7	$2 \rightarrow 2$	4×2^4	0.1	62	33	2:33	5
	$2 \rightarrow 3$	3×2^4	0.1	62	47	2:20; 3:27	6
	$3 \rightarrow 3$	3×3^3	0.2	84	54	2:13; 3:41	8
	$2 \rightarrow 4$	2×2^4	0.1	58	41	2:13; 4:28	6
	$4 \rightarrow 4$	2×4^2	0.1	**48**	41	2:2; 4:39	6
serp8	$2 \rightarrow 2$	4×2^4	0.1	68	39	2:39	6
	$2 \rightarrow 3$	3×2^4	0.2	66	50	2:19; 3:31	5
	$3 \rightarrow 3$	3×3^3	0.2	88	58	2:6; 3:52	8
	$2 \rightarrow 4$	2×2^4	0.1	54	41	2:10; 4:31	6
	$4 \rightarrow 4$	2×4^2	0.1	**48**	38	2:2; 4:36	6

Table 5.7: Component level synthesis results (continued)

circuit	radix	truth vectors, number and length	runtime, sec	component level generic parameters			
				sw. wires	operations		crit. path, comp-s
					total	by radix	
aes	$2 \rightarrow 2$	8×2^8	46.7	2450	1233	2:1233	14
	$2 \rightarrow 3$	6×2^8	55.3	2486	1517	2:455; 3:1062	14
	$3 \rightarrow 3$	6×3^6	397.2	2570	1529	2:46; 3:1483	15
	$2 \rightarrow 4$	4×2^8	31.7	1974	1340	2:300; 4:1040	14
	$4 \rightarrow 4$	4×4^4	27.8	**1954**	1326	2:14; 4:1312	12
misty7	$2 \rightarrow 2$	7×2^7	2.7	**248**	127	2:127	6
	$2 \rightarrow 3$	5×2^7	7.8	1100	669	2:188; 3:481	12
	$3 \rightarrow 3$	5×3^5	23.2	1182	712	2:32; 3:680	13
	$2 \rightarrow 4$	4×2^7	2.1	250	157	2:68; 4:89	7
	$4 \rightarrow 4$	4×4^4	10.4	512	355	2:3; 4:352	8
kasumi7	$2 \rightarrow 2$	7×2^7	2.7	252	130	2:130	6
	$2 \rightarrow 3$	5×2^7	7.9	1084	658	2:186; 3:472	12
	$3 \rightarrow 3$	5×3^5	22.8	1048	637	2:32; 3:605	13
	$2 \rightarrow 4$	4×2^7	2.0	**236**	154	2:64; 4:90	7
	$4 \rightarrow 4$	4×4^4	11.1	512	357	2:3; 4:354	8
misty9	$2 \rightarrow 2$	9×2^9	155.9	268	141	2:141	5
	$2 \rightarrow 3$	6×2^9	292.7	4776	2847	2:777; 3:2070	16
	$3 \rightarrow 3$	6×3^6	509.5	4948	2948	2:54; 3:2894	16
	$2 \rightarrow 4$	5×2^9	89.7	**250**	168	2:53; 4:115	6
	$4 \rightarrow 4$	5×4^5	526.2	456	320	2:4; 4:316	7
kasumi9	$2 \rightarrow 2$	9×2^9	156.7	256	132	2:132	5
	$2 \rightarrow 3$	6×2^9	287.9	4950	2946	2:768; 3:2178	15
	$3 \rightarrow 3$	6×3^6	509.8	5140	3044	2:50; 3:2994	16
	$2 \rightarrow 4$	5×2^9	90.3	**244**	165	2:55; 4:110	6
	$4 \rightarrow 4$	5×4^5	545.4	432	306	2:5; 4:301	7

Table 5.8: Technology level synthesis results

circuit	radix	Dynamic logic		Faraday 90nm (relaxed balancing)		AMS 0.35μm (relaxed balancing)		Faraday 90nm *single-rail*	
		sw. en., 10^{-3}pJ	area, c. units	sw. en., 10^{-3}pJ	area, c. units	sw. en., pJ	area, μm^2	sw. en.*, 10^{-3}pJ	area, μm^2
des1	$2 \to 2$	402.9	**2145**	**515.1**	**3300**	61.4	**44682**	**719.1**	**1470**
	$2 \to 3$	460.7	2843	607.6	5792	–	–	–	–
	$2 \to 4$	**305.2**	2551	563.8	6744	**48.4**	81622	1015.4	1984
	$4 \to 4$	329.0	3728	706.1	9857	49.8	116933	1645.3	12551
des2	$2 \to 2$	312.4	**1664**	**393.5**	**2452**	47.9	**33064**	**538.4**	**1125**
	$2 \to 3$	448.9	2819	606.3	6242	–	–	–	–
	$2 \to 4$	**277.5**	2282	499.0	5746	**44.3**	72094	880.0	1745
	$4 \to 4$	339.5	3848	735.1	10308	51.5	122352	1782.8	14078
des3	$2 \to 2$	307.5	**1638**	**387.4**	**2414**	**47.2**	**32552**	**523.2**	**1098**
	$2 \to 3$	473.2	2808	604.6	5162	–	–	–	–
	$2 \to 4$	**304.4**	2439	545.2	6236	48.5	76334	950.8	1884
	$4 \to 4$	332.0	3781	722.9	10171	50.3	120707	1743.2	14427
des4	$2 \to 2$	415.1	**2210**	**527.9**	**3314**	63.3	**45740**	**749.8**	**1501**
	$2 \to 3$	432.9	2670	574.9	5506	–	–	–	–
	$2 \to 4$	**350.4**	3037	631.7	7148	**56.0**	103570	1220.3	2372
	$4 \to 4$	375.0	4210	822.4	11486	57.3	135882	1870.6	13403
des5	$2 \to 2$	390.5	**2080**	**493.3**	**3090**	57.9	**42748**	**732.9**	**1462**
	$2 \to 3$	455.5	2802	619.3	6166	–	–	–	–
	$2 \to 4$	**300.5**	2567	579.9	6782	**48.1**	86772	1084.8	2113
	$4 \to 4$	349.6	3956	741.1	10271	53.2	128210	1849.2	15050
des6	$2 \to 2$	344.1	**1833**	**433.1**	**2694**	52.8	**36318**	**623.7**	**1286**
	$2 \to 3$	410.3	2538	547.2	5302	–	–	–	–
	$2 \to 4$	**314.0**	2463	545.4	6110	**49.8**	74214	948.2	1885
	$4 \to 4$	348.7	3932	742.9	10224	53.1	127922	1848.1	15043
des7	$2 \to 2$	329.5	**1755**	**416.4**	**2610**	50.5	**35226**	**598.6**	**1197**
	$2 \to 3$	434.2	2705	580.4	5710	–	–	–	–
	$2 \to 4$	**285.7**	2461	560.9	6570	**46.5**	90268	1127.1	2181
	$4 \to 4$	342.4	3887	727.1	10150	51.8	120666	1714.2	13109
des8	$2 \to 2$	310.1	**1651**	**395.8**	**2528**	46.5	**33998**	**547.9**	**1118**
	$2 \to 3$	453.4	2783	602.1	5800	–	–	–	–
	$2 \to 4$	**251.0**	2167	484.7	5988	**39.9**	72076	879.3	1707
	$4 \to 4$	311.7	3506	686.1	9642	47.5	114310	1631.0	12925

* The discussion on how to compare switching energies of balanced and single-rail implementations is given on page 115.

Table 5.9: Technology level synthesis results (continued)

circuit	radix	Dynamic logic		Faraday 90nm (relaxed balancing)		AMS 0.35µm (relaxed balancing)		Faraday 90nm *single-rail*	
		sw. en., 10^{-3}pJ	area, c. units	sw. en., 10^{-3}pJ	area, c. units	sw. en., pJ	area, µm^2	sw. en., 10^{-3}pJ	area, µm^2
serp1	$2 \to 2$	68.4	**364**	**88.7**	**582**	10.3	**7908**	128.3	**258**
	$2 \to 3$	88.5	519	112.0	988	–	–	–	–
	$2 \to 4$	**64.4**	520	106.5	1256	**9.7**	17018	**209.2**	408
	$4 \to 4$	78.7	843	163.5	2246	11.8	26778	374.8	2427
serp2	$2 \to 2$	80.6	**429**	**104.3**	**682**	11.8	**9080**	150.5	**299**
	$2 \to 3$	98.9	602	128.8	1218	–	–	–	–
	$2 \to 4$	**64.7**	576	142.3	1830	**10.5**	21618	**277.6**	538
	$4 \to 4$	81.2	870	163.6	2111	12.3	25558	406.1	2484
serp3	$2 \to 2$	73.2	**390**	**92.5**	**580**	10.8	**7942**	148.8	**295**
	$2 \to 3$	96.9	599	127.1	1198	–	–	–	–
	$2 \to 4$	**62.1**	535	127.5	1590	**10.0**	18876	**255**	491
	$4 \to 4$	68.6	735	130.4	1764	10.3	23554	348.9	2380
serp4	$2 \to 2$	75.7	**403**	**97.0**	**624**	11.5	**8750**	143.5	**289**
	$2 \to 3$	79.2	493	104.4	914	–	–	–	–
	$2 \to 4$	**64.8**	590	133.8	1598	10.4	20928	**256.7**	499
	$4 \to 4$	65.6	696	137.9	1873	**9.9**	22165	331.8	1920
serp5	$2 \to 2$	80.5	**429**	**101.4**	**632**	12.2	**9114**	146.2	**300**
	$2 \to 3$	80.3	471	101.9	816	–	–	–	–
	$2 \to 4$	**64.7**	576	122.7	1542	10.2	18698	**232.4**	452
	$4 \to 4$	59.7	618	122.8	1664	**9.0**	19828	314.4	2310
serp6	$2 \to 2$	80.5	**429**	**99.2**	**628**	12.3	**8966**	154.3	**309**
	$2 \to 3$	100.5	579	123.3	976	–	–	–	–
	$2 \to 4$	**64.6**	562	125.7	1430	10.4	19972	**248**	485
	$4 \to 4$	72.2	744	131.1	1786	**9.7**	21438	338.8	2361
serp7	$2 \to 2$	75.8	**403**	**98.7**	**654**	11.4	**8898**	145.1	**289**
	$2 \to 3$	78.5	478	96.5	796	–	–	–	–
	$2 \to 4$	69.5	588	124.6	1494	11.2	18520	**227**	442
	$4 \to 4$	**65.2**	684	145.3	1980	**9.9**	23188	325.6	1480
serp8	$2 \to 2$	83.0	**442**	**105.9**	**676**	12.6	**9296**	160.7	**326**
	$2 \to 3$	81.2	496	106.1	982	–	–	–	–
	$2 \to 4$	67.2	603	125.3	1506	10.8	21110	**255**	495
	$4 \to 4$	**65.6**	696	128.1	1662	**9.7**	20997	300.5	1863

Table 5.10: Technology level synthesis results (continued)

circuit	radix	Dynamic logic		Faraday 90nm (relaxed balancing)		AMS 0.35µm (relaxed balancing)		Faraday 90nm *single-rail*	
		sw. en., 10^{-3}pJ	area, c. units	sw. en., 10^{-3}pJ	area, c. units	sw. en., pJ	area, µm^2	sw. en., 10^{-3}pJ	area, µm^2
aes	$2 \to 2$	2996.3	**15925**	**3962.3**	**26940**	448.4	**367842**	6000.9	**11620**
	$2 \to 3$	3190.0	20139	4234.5	42670	–	–	–	–
	$2 \to 4$	**2415.6**	21893	5002.3	64692	**384.2**	774626	**8866.2**	16936
	$4 \to 4$	2541.2	28672	6253.3	89620	398.8	1038716	12946.1	80834
misty7	$2 \to 2$	302.8	**1612**	**386.3**	**2466**	**46.2**	**33372**	**535.1**	**1096**
	$2 \to 3$	1414.8	8969	1915.2	18586	–	–	–	–
	$2 \to 4$	**309.2**	2423	520.0	5778	48.5	71200	870.4	1719
	$4 \to 4$	678.0	7686	1628.8	23427	105.4	276103	3705	29171
kasumi7	$2 \to 2$	307.6	**1638**	**391.9**	**2494**	46.9	**33736**	541.6	**1113**
	$2 \to 3$	1393.9	8830	1904.1	19870	–	–	–	–
	$2 \to 4$	**291.9**	2304	488.4	5442	**45.8**	67062	**843.5**	1669
	$4 \to 4$	680.2	7746	1610.9	23159	105.2	273467	3805.8	31070
misty9	$2 \to 2$	327.5	**1742**	**425.5**	**2806**	49.5	**38152**	620.5	**1227**
	$2 \to 3$	6147.3	39087	8413.4	88316	–	–	–	–
	$2 \to 4$	**310.6**	2647	577.3	7032	**48.7**	85708	**1035.7**	1997
	$4 \to 4$	596.1	6564	1527.6	22108	94.4	259196	3298.5	23223
kasumi9	$2 \to 2$	312.9	**1664**	**408.2**	**2712**	**47.2**	**36912**	**597.9**	**1184**
	$2 \to 3$	6379.7	40739	8745.1	92594	–	–	–	–
	$2 \to 4$	**302.9**	2524	563.2	6756	47.6	82154	1008.7	1947
	$4 \to 4$	565.8	6240	1442.6	20872	89.4	244852	3172.4	22987

Table 5.11: Comparison between conversion and synthesis results

circuit	switching wires			
	Reed-Muller synthesis		converted from single-rail	
	binary (dual-rail)	mixed radix (dual-rail, 1-of-4)	binary (dual-rail)	mixed radix (dual-rail, 1-of-4)
2-bit adder	44	46	20	14
4-bit multiplier	112	110	56	58
Kasumi S-box 7	252	236	250	264
Kasumi S-box 9	256	244	256	264
AES S-box	2450	1974	1594	1640

indicates that mathematically the functions are not perfectly bound to the quaternary radix. The mixed radix approach replaces parts of the circuit with lower radix logic. The search through all possible polarity forms provides the optimal balance between the radices. From this point of view the conversion approach, discussed in Chapter 3, may have greater flexibility as it considers the radix of each particular gate. But since it works at the gate level, it requires explicit conversion of signals, implying a huge amount of additional logic (up to 35% of the circuit). Hence, in practice the radix conversion is less efficient than mixed radix synthesis.

At the physical level, the difference in energy costs of the GF components changes the picture even more; this is why simple counting of operations does not always serve as a sufficient criterion for the search.

Let's consider dynamic logic benchmarks first. The switching energy of gates in binary-to-quaternary circuits is 7–42% less than in any other radix combination, even if the component-level statistics are better for uniform quaternary. However the area is bigger than in binary. Since one-hot encoded higher radix signals of radix q have q wires per signal, the amount of logic required to maintain a computation increases with the radix. Hence it is a property of the encoding that, despite reduced switching energy, higher radix logic has a larger area. So here we have a power-area trade-off: comparing binary-to-quaternary circuits with purely binary, one can observe a 15–25% switching energy reduction for the cost of 15–30% area overhead. The benefit of mixed-radix RM expansions here is in providing such a trade-off. Uniform radix circuits show up to 18%

energy saving (versus binary) for 34–57% more area, and in some benchmarks there is no energy saving at all.

For Faraday 90nm implemented circuits higher radices have a totally inefficient outcome as was expected from Table 5.2, and even binary circuits have worse characteristics than their dynamic logic counterparts. Consequently one could conclude that efficient MVL synthesis requires custom design components, however it is important to remember that the relation of physical characteristics to radices may change from library to library. The benchmarks built using AMS 0.35μm library have different radix efficiency. The results for AES show 14% less switching energy in the binary-to-quaternary circuit than in the purely binary one. Hence we cannot talk about global efficiency or inefficiency; the approach provides a range of solutions, so the designer may choose the most appropriate. In our benchmarks for the considered 90nm technology the results show that custom design circuits are a lot more efficient than RTL-implemented ones.

Table 5.11 compares the approaches presented in Chapter 3 and Chapter 4. Although the conversion driven mixed radix design is considered to be inefficient due to the large amount of signal conversion logic added, it is still more efficient than Reed-Muller synthesis. There are two possible reasons: 1) fixed polarity Reed-Muller expansion is not an appropriate form for implementing these functions; 2) RMMixed tool's logic minimisation provides insufficient optimisation and needs to be improved.

5.5 Summary

Muti-valued and mixed radix Reed-Muller expansions, described in Chapters 2 and 4, have been applied to the security aware logic synthesis. The implementation of power-balanced Galois field arithmetic components has been provided for RTL and custom design dynamic logic. The theoretical background has been implemented in a tool. A number of related optimisations have been also introduced, however the current version of the tool is still work-in-progress. Better logic minimisation algorithms and

reduced computation complexity are still required.

Multiple security related benchmarks have been synthesised in different radices and technologies. The set of tested radix models include uniform radix functions and some two-level radix models. Mixed radix domain $\{2, 4\} \rightarrow 4$ model has been found insufficiently beneficial in comparison with the optimisation possibilities of other radix models and has been rejected. The binary-to-quaternary model has been found more efficient in many cases, however there is no objective general trends as each function has different relation to radices. The main contribution of the mixed radix approach is that it provides a range of solutions, so the designer may choose the most appropriate.

In our benchmarks, for the considered 90nm technology, the results show that custom design circuits are a lot more efficient for security application than RTL-implemented ones.

Section 3.2.5 has been previously published in [56] for RTL-based component implementations and in [16] for dynamic logic implementations; it is also accepted for publication in [53]. Section 5.3 has been published in [58]. [53] includes the benchmark results and related discussion from Section 5.4. Workflow example, presented in Section A.5, has been published in [57].

Chapter 6

Conclusion

M-of-n codes in asynchronous circuits have been considered as an efficient countermeasure to side-channel attacks, particularly to power analysis. In scope of these techniques a number of existing design environments and synthesis tools have been analysed for their capabilities.

Two design approaches have been determined and elaborated into design flows. The conversion approach suggests reusing the existing insecure designs in order to produce secure higher radix circuits. The synthesis approach uses multi-valued logic synthesis theory. The idea of using mixed radices has been applied to both approaches.

For the conversion driven design, the results have shown a number disadvantages with respect to different optimisation criteria. The tests revealed power and area overhead, mostly caused by additional logic required to connect lower radix parts with higher radix parts. The number of signal converters in the circuit can be up to 35% of the total number of encoded components. However, the advantageous property of the developed CDD algorithms is a considerably smaller computational cost in comparison with the developing a completely new synthesis; and the approach more easily fits within the standard EDA flow. From this point of view, Verimap tool [67] is recommended for conversion driven design, as it applies secure dual-rail encoding without converting the circuit radix.

The synthesis approach is based on the developed theory of mixed radix Reed-

Muller expansions and cross-radix computation at the level of mathematical representation. Compared to the existing radix-conversion approach, the presented method avoids using signal conversion, producing more efficient results. The overall benefit of this theory is that it provides a new dimension for logic/interconnect trade-off, and allows the designer to observe the effect of changing the radix in the whole range of radix combinations.

A flow based on mixed and uniform radix RM expansions has been proposed and implemented within a tool. The tool has been used in a real-life application example – secure implementations of cryptographic algorithms – in order to compare the efficiency of security-applied custom design logic and RTL-based MVL implementations. Circuits based on dynamic logic 1-of-n encoded components use less switching energy in binary-to-quaternary compared to any other radix, and also reduce the switching activity of the wires. The price is area overhead. However, if using uniform radices only, the same energy gain can be achieved for almost double overhead. RTL implementations are significantly worse than the dynamic logic benchmarks, but simplier to implement and finalise using the standard tools.

The use of mixed radix also contributes advantageously to security due to its intrinsic diffusion of binary values between signals representing a mixed radix value. A data signal passing through the mixed radix circuit is "unpredictably" reformed due to the mixing with other signals and splitting again for multiple times. Therefore a tracing of the initial data becomes more complex, thereby increasing the resistance to DPA attacks.

In general, the thesis presents a novel view on using radices: different radices can be combined together within a single circuit or even a single function in order expand the search space for optimisation. The flow using Reed-Muller expansions adds new features to the design automation enabling MVL synthesis that is flexible to application requirements. Power balancing has been implemented providing security properties for the price of increased power consumption and area. The proposed flow has been applied in a typical example of a cryptographic algorithm (DES), and has been

recognised to be easy to follow and acceptable for the industrial use. Previously this method has never been used in a practically applied design flow.

6.1 Future work

The theory of mixed radix Reed-Muller expansions is complete, but some improvement is still required for the synthesis tool.

Runtime Currently the tool computes Reed-Muller expansions using Green's direct method and derived mixed radix solutions. For some radix q and n input variables, this implies the multiplication of q^n-by-q^n matrix and q^n-long vector over GF(q), which is a very high computational complexity. Presumably, the improvement of the Reed-Muller computation algorithm can be derived from the previously known optimisation techniques for uniform radix Reed-Muller expansions [24, 59, 36, 25]. The suggestion is to find the possibility of extending these algorithms to two-level radix models.

In addition, it is essential to optimise the tool for multicore processors. This is relatively straightforward, since each polarity can be computed concurrently.

Input formats Input in a truth vector format is natuaral for Green's direct method of computation, however it is absolutely not user friendly. The planned new feature of the tool is the support of BLIF-MV format [42]. It would also be very helpful to have the possibility of reading binary BLIF and Verilog files and mapping the specification into higher radices. This may imply the radix conversion similar to the one described in Chapter 3, but applied at the per-module scale rather than at the gate level. The latter issue requires a dedicated research.

Logic minimisation As can be observed from Table 5.4 in comparison with Tables 5.5–5.7, the improved logic minimisation algorithms significantly increased the efficiency of the synthesis. However, comparing the results with the dual-rail circuits

obtained using the conversion approach, most of the converted binary circuits are still better, because the original circuits have been synthesised by the mature EDA tools, such as design compiler. These tools certainly outperform our tool, therefore it needs more work on the optimisation part in order become a viable solution for EDA.

Application The security application was the main goal of the presented research. However, the proposed methodology is not limited to this particular application area. Exploration of other encodings and possible application areas might uncover more advantages of the tool and the design flow.

Appendix A

RMMixed User Manual

A.1 Tool features overview

The tool is implemented in two versions: command line tool (`rmmixed-cmd.jar`) and Tcl console (`rmmixed.jar`). Both share the same internal structure and algorithms, the different is in the interface.

Command line tool can be used to compute a Reed-Muller expansions of one of the predefined radix models, which include uniform radix binary, ternary, quaternary expansions and the following two-level radix modes: $2 \rightarrow 3, 2 \rightarrow 4, 4 \rightarrow 2$. The expansion is minimised and mapped into provided library of Galois field arithmetic components. The circuit optimisation can be made by estimating total energy, area or switching activity of the circuit. At the moment the only accepted input format is the truth table. Supported output format is structural Verilog. For the command line options refer to Section A.2.

Tcl version of the tool is designed mainly for reseach purposes, while the command line tool has been made for intensive benchmark runs. Tcl version has an extended list of features accessible through the Tcl commands:

· the tool supports arithmetic in any Galois field;

· Reed-Muller expansions can be computed in any uniform radix (assuming that

corresponding GF exists) or any radix model described in Chapter 4;

- the tool includes an expression calculator for GF arithmetic, see Section A.3.

The complete list of Tcl command is given in Section A.4.

A.2 Command line tool: rmmixed-cmd

Synopsis

```
java -jar rmmixed-cmd.jar ?-lib file_name?  ?-in file_name?  ?-out
file_name?  ?other_options?
```

Options

-in *file_name*

Read truth vectors from the specified file. By default, reads from stdin.

-out *file_name*

Write output to the specified file. By default, writes to stdout.

-lib *file_name*

Read GF component library.

-radix-binary or -rbb

Synthesise uniform radix binary expansions.

-radix-ternary or -rtt

Synthesise uniform radix ternary expansions.

-radix-quaternary or -rqq

Synthesise uniform radix quaternary expansions.

-radix-binary-to-ternary or -rbt

Synthesise using $2 \rightarrow 3$ radix model.

-radix-binary-quaternary or -rbq

Synthesise using $2 \rightarrow 4$ radix model.

-radix-quaternary-to-binary or -rqb

Synthesise using $4 \rightarrow 2$ radix model.

-polarity *k* or -p *k*

Do not search for the best polarity, synthesise using the specified k.

-zero-polarity or -p0

Do not search for the best polarity, synthesise using $k = 0$.

-optimise-switches or -ow

Optimise the circuit using the number of switching wires as a search criterion.

-optimise-energy or -oe

Optimise the circuit using the estimated switching energy as a search criterion.

-optimise-area or -oa

Optimise the circuit using the total area of gates as a search criterion.

-optimise-timing or -ot

Optimise the circuit using the length of the longest path (maximum delay) as a search criterion.

-name *module_name*

Assign a module name to the output circuit.

-interface-binary or -ib

Frame the mixed radix circuit with the signal conversion logic, if necessary, in order to make all ports binary.

-interface-ternary or -it

Frame the mixed radix circuit with the signal conversion logic, if necessary, in order to make all ports ternary.

-interface-quaternary or -iq

Frame the mixed radix circuit with the signal conversion logic, if necessary, in order to make all ports quaternary.

Examples

```
java -jar rmmixed-cmd.jar -lib "vlib/gflib_dynamic.v" -in "serp1_q.in"
-out "serp1.v" -rbq -name SerpSBox1 -oe
```

Reads GF library of dynamic logic components and a quaternary specification of the Serpent S-box 1, computes binary-to-quaternary Reed-Muller expansion optimised by switching energy, and writes the circuit into a verilog file.

```
java -jar rmmixed-cmd.jar -lib "vlib/gflib_relaxed_generic.v" -in
"aes_b.in" -out "aes.v" -rbb -name AES -ow -p0
```

Reads GF library of relaxed implementations using generic cells and a binary specification of the AES S-box, computes zero polarity binary Reed-Muller expansion optimised by the number of switching wires, and writes the circuit into a verilog file.

For a complete design flow example see Section A.5.

A.3 Galois field expression calculator

GF expression can be calculated using `gfexpr` Tcl command. It accepts the following types of data:

- GF value. GF constants can be expressed by integer numbers and uppercase latin letters.
- Matrices matrices of GF elements. Matrices can be created using `matrix` Tcl command.
- Polynomials of transcendental variable x. Polynomials can be created using `poly` Tcl command.
- Integer numbers (to express the powers).

The expression string can contain nested Tcl commands or Tcl variables.

Table A.1 displays the list of supported operators.

Table A.1: `gfexpr` operators

operator	type	description
unary operators		
-	`-value`	negate GF value, $(-x)$
	`-matrix`	negate each element of the matrix
~	`~value`	get inverse GF value, x^{-1}
	`~matrix`	invert matrix
binary operators		
+	`value+value`	add in GF, $x + y$
	`element+poly` `poly+element`	treat GF element as a polynomial and add the polynomials
	`poly+poly`	add polynomials
	`matrix+matrix`	add matrices
-	`value-value`	subtractin GF, $x + (-y)$
	`element-poly` `poly-element`	treat GF element as a polynomial and subtract the polynomials
	`poly-poly`	subtract polynomials
	`matrix-matrix`	subtract matrices
*	`value*value`	multiply in GF, $x \cdot y$
	`value*poly` `poly*value`	treat GF element as a polynomial and multiply the polynomials
	`poly*poly`	multiply polynomials
	`value*matrix` `matrix*value`	multiply matrix by value
	`matrix*matrix`	multiply matrices
/	`value/value`	divide GF values, $x \cdot y^{-1}$
	`value/poly` `poly/value`	treat GF element as a polynomial and divide the polynomials
	`poly/poly`	divide polynomials
%	`value%value`	always 0
	`value%poly` `poly%value`	treat GF element as a polynomial and find the remainder of division
	`poly%poly`	find remainder of the polynomial division
**	`matrix**matrix`	Kronecker product of two matrices
^	`value^integer`	exponentiate GF value, x^n
	`poly^integer`	exponentiate polynomial
( )		change operator precdence

Examples

```
gfexpr -gf [gf 4] A*(B-1)
```

Calculates $A\,(B-1)$ over GF(4).

```
set_default_gf [gf 2]
set p1 [poly 101]
set p2 [poly 11]
gfexpr $p1*$p2
```

Multiplies polynimials of transcendental x: $\left(x^2+1\right)(x+1)$ over GF(2).

```
rm_radix_model -in 4 -out 4
set_default_gf [gf 4]
set d [vector BBAABB1ABBAABB1A]
set s0 [rm_get_smatrix 0]
set w0 [gfexpr ~$s0]
gfexpr ($w0**$w0)*d
```

Manually compute zero-polarity quaternary Reed-Muller expansion.

A.4 List of Tcl commands

get_primitive_poly Get primitive poly for the specified prime power Galois field.

gf Create Galois field for the specified characteristic.

gfexpr Compute the expression in the Galois field.

gfpp Create prime power Galois field for the specified primitive polynomial.

help Get help on Tcl commands.

identity_matrix Create an identity matrix.

is_prime_poly Tells if the specified polynomial is irreducible.

is_primitive_poly Tells if the specified polynomial is a primitive polynomial.

list_prime_polys Create the list of irreducible polynomials of the specified degree.

matrix Create a matrix parsing the list of strings.

poly Create polynomial using the coefficients provided.

print_gf_elements Print elements of a Galois field.

print_gf_operations Print operation truth tables for a Galois field.

print_poly Convert polynomial object into a readable string.

print_prime_polys Print the list of irreducible polynomials into a string.

rm Solve RM expansions.

rm_get_smatrix Get constant power matrix for the specified polarity in the preset radix model.

rm_get_wmatrix Get Green's matrix for the specified polarity in the preset radix model.

rm_radix_model Create new Reed-Muller synthesis configuration.

rm_reset_spec Reset all specification tables for Reed-Muller synthesis.

rm_synthesise Synthesise Reed-Muller expansions.

set_default_gf Set default Galois field for other Tcl commands.

set_format_elements Set output format for the elements of prime power fields.

spec_add_vector Add a truth vector to the specification.

spec_library Set GF component library and optimisation criteria.

spec_print_ports Print list of inputs and outputs in the current circuit specification.

spec_read Read truth vectors from file.

vector Create a single row matrix parsing the string of values.

get_primitive_poly

Name

get_primitive_poly - Get primitive poly for the specified prime power Galois field.

Synopsis

```
get_primitive_poly ?-gf gf?
```

Description

get_primitive_poly returns the characteristic irreducible polynomial that has been used to create the specified prime power Galois field. If `$p` is some prime polynomial and `$gf` is a field created using `set gf [gfpp $p]` command, `getprimitivepoly $gf` will return `$p`.

Options

-gf *gf*

Galois field (Java reflection object), optional. Prime power Galois field. If the parameter is omitted, the field previously set by `set_default_gf` is used.

Returns

Polynomial (Java reflection object).

Examples

```
get_primitive_poly -gf 8
```

Displays the default polynomial, used to create GF(8), which is $x^3 + x + 1$.

gf

Name

gf - Create Galois field for the specified characteristic.

Synopsis

```
gf order
```

Description

gf creates Galois field for the specified *order*. The command caches the result, so the consequent calls do not recompute operation tables.

gf always returns a Galois field for a valid *order*. For nonprime fields, if there is no cached resut previously created by `gfpp` command, the first primitive polynomial is taken to create a new field. For example, for some $q = p^t$, `gf $q` can replace the following code:

```
foreach prime [list_prime_polys -gf [gf $p] $t] {
  if { ![is_primitive_poly $prime] } then { continue }
  return [gfpp $prime]
}
```

You can always use `get_primitive_poly` to figure out which polynomial has been used to create the field.

Note: GF(q) is isomorphic to any other GF(q) regardless on the primitive polynomial.

Options

order

Integer. Order of the Galois field, must be a power of prime.

Returns

Galois field (Java reflection object).

Examples

`gf 3`

Creates GF(3).

```
gfpp [poly -gf [gf 3] {122} ]
print_poly [get_primitive_poly -gf [gf 9]]
```

First line creates GF($3^2 = 9$) with the primitive polynomial $x^2 + 2x + 2$ over GF(3). The second line retrieves the previously created GF(9) and prints it's primitive polynomial. Without the first line `gf 9` would use default polynomial $x^2 + x + 2$ over GF(3) to create a new field.

```
print_gf_operations -gf [gf 4]
```

Creates GF(4) and prints all operation tables. Since there is only one irreducible polynomial of degree 2 over GF(2), `gf 4` successfully replaces `gfpp [poly -gf [gf 2] 111]`.

gfexpr

Name

gfexpr - Compute the expression in the Galois field.

Synopsis

```
gfexpr ?-gf gf?  expression
```

Description

gfexpr is a powerful tool to perform computations in Galois fields. See Section A.3.

Options

-gf *gf*

Galois field (Java reflection object), optional. Galois field, over which the expression is computed. If the parameter is omitted, the field previously set by `set_default_gf` is used.

expression

String. Arithmetic expression.

Returns

Result of the expression can be a GF value (String), a polynomial or a matrix (Java reflection objects).

gfpp

Name

gfpp - Create prime power Galois field for the specified primitive polynomial.

Synopsis

```
gfpp polynomial
```

Description

The characteristic of the Galois field is determined by the characteristic of prime field used to create the polynomial and the degree of the polynomial, i.e. $a_n x^n + \ldots + a_1 x + a_0$ over GF(p) will create GF(p^n). Both prime Galois field and polynomial degree can be specified when creating the polynomial using `poly` command.

You can use `is_prime_poly` and `is_primitive_poly` to check if the polynomial can be used to create a field. `list_prime_polys` lists all prime polynomials of the given degree for the specified prime Galois field.

Options

polynomial

Polynomial (Java reflection object). Irreducible polynomial over GF prime. Polynomial can be created using `poly` command.

Returns

Galois field (Java reflection object).

Examples

```
gfpp [poly -gf [gf 2] {1101} ]
```

Creates GF($2^3 = 8$) with primitive polynomial $x^3 + x^2 + 1$.

help

Name

help - Get help on Tcl commands.

Synopsis

```
help ?command?
```

Options

command

String. Tcl command to get help for.

Returns

String.

identity_matrix

Name

identity_matrix - Create an identity matrix.

Synopsis

```
identity_matrix ?-gf gf?  size
```

Options

-gf *gf*

Galois field (Java reflection object), optional. Galois field, in which to create the matrix. If the parameter is omitted, the field previously set by `set_default_gf` is used.

size

Integer. Size of the matrix.

Returns

Matrix (Java reflection object).

Examples

```
identity_matrix -gf [gf 2] 4
```

Creates 4-by-4 identity matrix over GF(2).

is_prime_poly

Name

is_prime_poly - Tells if the specified polynomial is irreducible.

Synopsis

```
is_prime_poly poly
```

Options

poly

Polynomial (Java reflection object). The polynomial to check.

Returns

Integer. 1 if the polynomial is irreducible, 0 otherwise.

is_primitive_poly

Name

is_primitive_poly - Tells if the specified polynomial is a primitive polynomial.

Synopsis

```
is_primitive_poly poly
```

Options

poly

Polynomial (Java reflection object). The polynomial to check.

Returns

Integer. 1 if the polynomial is primitive, 0 otherwise.

list_prime_polys

Name

list_prime_polys - Create the list of irreducible polynomials of the specified degree.

Synopsis

```
list_prime_polys ?-gf gf?  degree
```

matrix

Name

matrix - Create a matrix parsing the list of strings.

Synopsis

```
matrix ?-gf gf?  rows
```

Options

-gf *gf*

Galois field (Java reflection object), optional. Galois field, in which to create the matrix. If the parameter is omitted, the field previously set by `set_default_gf` is used.

rows

List. List of strings, each one contains the elements of a matrix row.

Returns

Matrix (Java reflection object).

Examples

```
identity_matrix -gf [gf 2] 4
```

Creates 4-by-4 identity matrix over GF(2).

poly

Name

poly - Create polynomial using the coefficients provided.

Synopsis

```
poly ?-gf gf?  coefficients
```

Options

-gf *gf*

Galois field (Java reflection object), optional. Galois field, over which the polynomial is computed. If the parameter is omitted, the field previously set by `set_default_gf` is used.

coefficients

String. Polynomial coefficients in a form of a vector.

Returns

Polynomial (Java reflection object).

Examples

```
poly -gf [gf 2] 111
```

Creates polynomial $x^2 + x + 1$ over GF(2).

print_gf_elements

Name

print_gf_elements - Print elements of a Galois field.

Synopsis

```
print_gf_elements ?-gf gf?
```

Options

-gf *gf*

Galois field (Java reflection object), optional. If the parameter is omitted, the field previously set by `set_default_gf` is used.

Returns

String.

print_gf_operations

Name

print_gf_operations - Print operation truth tables for a Galois field.

Synopsis

```
print_gf_operations ?-gf gf?
```

Description

print_gf_operations prints truth tables for GF operations, including addition, multiplication, inverse, and exponentiation.

Options

-gf *gf*

Galois field (Java reflection object), optional. Galois field. If the parameter is omitted, the field previously set by `set_default_gf` is used.

Returns

String.

print_poly

Name

print_poly - Convert polynomial object into a readable string.

Synopsis

```
print_poly poly
```

Options

poly

Polynomial (Java reflection object).

Returns

String.

print_prime_polys

Name

print_prime_polys - Print the list of irreducible polynomials into a string.

Synopsis

```
print_prime_polys ?-gf gf?  degree
```

Description

print_prime_polys is a convenience command for printing the list of irreducible polynomials. Thus, `printprimepolys -gf [gf $q] $n` replaces the following code:

```
set str ""
foreach prime [list_prime_polys -gf [gf $q] $n] {
  set str "$str[print_poly $prime]\n"
}
return $str
```

Options

-gf *gf*

Galois field (Java reflection object), optional. Galois field, over which the polynomials are computed. If the parameter is omitted, the field previously set by `set_default_gf` is used.

degree

Integer. Degree of the polynomials to list.

Returns

String. List of polynomials as a string.

Examples

```
print_prime_polys -gf [gf 2] 3
```

Prints all irreducible polynomials of degree 3 over GF(2).

rm_get_smatrix

Name

rm_get_smatrix - Get constant power matrix for the specified polarity in the preset radix model.

Synopsis

```
rm_get_smatrix polarity
```

Description

rm_get_smatrix returns S_k matrix for the specified k and the radix model previously set by `rm_radix_model` command.

Options

polarity

Integer.

Returns

Matrix (Java reflection object).

rm_get_wmatrix

Name

rm_get_wmatrix - Get Green's matrix for the specified polarity in the preset radix model.

Synopsis

```
rm_get_wmatrix polarity
```

Description

rm_get_wmatrix returns W_k matrix for the specified k and the radix model previously set by `rm_radix_model` command.

Options

polarity

Integer.

Returns

Matrix (Java reflection object).

rm_radix_model

Name

rm_radix_model - Create new Reed-Muller synthesis configuration.

Synopsis

```
rm_radix_model -in gfinput -out gfoutput
```

Description

rm_radix_model defines a two-level radix model for further Reed-Muller synthesis calls using `rm_synthesise` command. If input radix is equal to output radix, the uniform radix synthesis is applied.

Options

-in *gfinput*

Integer. Input radix

-out *gfoutput*

Integer. Output radix

rm_reset_spec

Name

rm_reset_spec - Reset all specification tables for Reed-Muller synthesis.

Synopsis

```
rm_reset_spec
```

rm_synthesise

Name

rm_synthesise - Synthesise Reed-Muller expansions.

Synopsis

```
rm_synthesise ?-polatity polarity?  ?-out file_name?
```

Description

rm_synthesise computes Reed-Muller expansion for the radix model previously set by `rm_radix_model` command, optimises it and maps it using the library loaded using `spec_library` command. The truth vectors can be specified using `spec_add_vector` command.

Options

-polarity *polarity*

Integer, optional. Polarity number. If not specified, the tool searches for the best one.

-out *file_name*

String, optional. Output file name. By default prints to console output.

set_default_gf

Name

set_default_gf - Set default Galois field for other Tcl commands.

Synopsis

```
set_default_gf ?gf?
```

Options

gf

Galois field (Java reflection object), optional. Default Galois field to be set. If not specified, the default value is reset.

set_format_elements

Name

set_format_elements - Set output format for the elements of prime power fields.

Synopsis

```
set_format_elements ?format?
```

Description

Supported element formats are:

`num` - integer numbers only in the order of the field elements.

`alnum` - (default) integers for prime field elements and uppercase letters for others.

`poly` - print as polynomials of transcendental x.

`vec` - print as a coefficient vectors of polynomials of transcendental x.

`vecls` - print as a coefficient vectors of polynomials of transcendental x, the least significant digit first.

Options

format

String, optional. One of the `num`, `alnum`, `poly`, `vec`, and `vecls` flags.

spec_add_vector

Name

spec_add_vector - Add a truth vector to the specification.

Synopsis

```
spec_add_vector vector
```

Options

vector

String, optional. Truth vector.

spec_library

Name

spec_library - Set GF component library and optimisation criteria.

Synopsis

```
spec_library ?-opt flags?  path
```

Description

Optimisation parameter may contain one of the following flags:

`w` - optimise the number of switching wires.

`e` - optimise switching energy.

`a` - optimise area.

`t` - optimise timing.

Options

-opt *flags*

Integer, optional. Optimisation flags.

path

String. Library file path.

spec_print_ports

Name

spec_print_ports - Print list of inputs and outputs in the current circuit specification.

Synopsis

```
spec_print_ports
```

spec_read

Name

spec_read - Read truth vectors from file.

Synopsis

```
spec_read path
```

vector

Name

vector - Create a single row matrix parsing the string of values.

Synopsis

```
vector ?-gf gf?  value
```

Description

Vector is a single row matrix.

Options

-gf *gf*

Galois field (Java reflection object), optional. Galois field, in which to create the vector. If the parameter is omitted, the field previously set by `set_default_gf` is used.

value

String. String of elements.

Returns

Matrix (Java reflection object).

A.5 Workflow example

We have chosen DES as a classic example of cryptographic algorithm since it is simple enough to fit into a short discussion. It was processed through the design flow illustrated in Figure 3.2. Basic commands are listed in Algorithm 3.

The first step of the design process is the synthesis of S-boxes using Reed-Muller expansions. Specified command line options mean that the synthesised modules have a 1-of-4 encoded quaternary external interface (*-iq*), a mixed radix approach is applied

Algorithm 3 DES design flow commands

```
java -jar rmmixed-cmd.jar -iq -rbq -ow -lib gflib_relaxed_generic.v -in des/des1_q -out
des1.v
...
java -jar rmmixed-cmd.jar -iq -rbq -ow -lib gflib_relaxed_generic.v -in des/des8_q -out
des8.v

analyze -format verilog gflib_relaxed_generic.v des1.v ...  des8.v descore.v
elaborate descore
compile -exact_map
write -format verilog -hierarchy -output descore_syn.v

htcomp -input des_control.ht
htmap -input des_control.hcl des
htlink -input descore_syn.v -top des_htmap.v des
htpost -input des_htlink.v des
sed 's/HDDFFPQ1/SpacerDFF/' des_htpost.v > des_htpost_sp.v
```

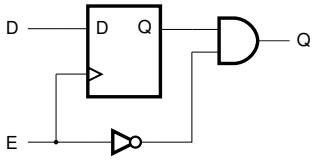


Figure A.1: Simple spacer injecting register

in a form of quaternary function of binary arguments (*-rbq*), i.e. $2 \rightarrow 4$ radix model is used, and the applied optimisation minimises wire switching (*-ow*). The complete list of command line options is given in Appendix A. A library of power-balanced components is provided in the file `gflib_relaxed_generic.v`.

After the modules have been presynthesised, Synopsys design compiler is used to elaborate all Verilog files into a single gate-level netlist. An important option at this stage is that we use exact technology mapping (*-exact_map*), otherwise compiler would reduce redundant logic paths used for power-balancing.

The next step is to process the design in a sequence of TiDE tools. Finally, in order to enforce spacers in the circuit, the UNIX stream editor (*sed*) replaces all occurrences of D flip-flops (HDDFFPQ1) with instances of the register SpacerDFF illustrated in Figure A.1. When the Enable signal from the handshake control unit is high, a spacer is generated; when Enable is low data is passed. The cycles are also asymmetric, so the spacer is generated for only a small percentage of the cycle (20 – 30%). The addition

of 20ps delay gate at the front of the register is trivial and does not affect the timing constraints. A big disadvantage of this approach is that flip-flops never get reset, i.e. they never capture the spacer value. Consequently, although they give balancing to the logic, they are not power-balanced themselves. However, compromised balancing can be accepted in order to reduce power consumption as long as the number of memory cells is significantly smaller than the number of logic gates. The number of replaced memory cells in our example is 5.3% of the total number of gates.

The final file `des_htpost_sp.v` contains an asynchronous power balanced m-of-n encoded circuit. Layout has not been applied in this example; this is a subject for future research.

Bibliography

[1] http://async.org.uk/sure/rmmixed.

[2] www.handshakesolutions.com.

[3] www.pulsic.com.

[4] Cadence Design Systems. www.cadence.com.

[5] Federal information processing standards FIPS 140-3 (draft). National Institute of Standards and Technology.

[6] Magma design automation. www.magma-da.com.

[7] Synopsys. www.synopsys.com.

[8] *3GPP Technical Specification 35.202*, 2001. v3.1.1.

[9] *Specification for the Advanced Encryption Standard (AES)*, 26 Nov 2001. Federal Information Processing Standards Publication 197.

[10] LM Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[11] M. Aigner, S. Mangard, F. Menichelli, R. Menicocci, M. Olivieri, T. Popp, G. Scotti, and A. Trifiletti. Side channel analysis resistant design flow. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pages 4 pp. –2912, 0-0 2006.

[12] R.J. Anderson, E. Biham, and L.R. Knudsenin. Serpent and smartcards. In *Cardis 98*, pages 257–264, 2000.

[13] Y. Baba, A. Miyamoto, N. Homma, and T. Aoki. Multiple-valued constant-power adder for cryptographic processors. In *Proc. of ISMVL '09.*, 2009.

[14] W.J. Bainbridge, W.B. Toms, D.A. Edwards, and S.B. Furber. Delay-insensitive, point-to-point interconnect using m-of-n codes. In *Proc. of ASYNC'03*, 2003.

[15] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. *Lecture Notes in Computer Science*, 1294:513–??, 1997.

[16] F. Burns, A. Bystrov, A. Koelmans, and A. Yakovlev. Design and security evaluation of balanced 1-of-n circuits. Technical report, School of EECE, Newcastle University, Mar 2010.

[17] A. Bystrov, D. Sokolov, A. Yakovlev, and A. Koelmans. Improving the security of dual-rail circuits. In *Proc to Cryptographic Hardware and Embedded Systems, CHES 2004*, pages 282–97, 2004.

[18] Jean-Sébastien Coron and Louis Goubin. On Boolean and arithmetic masking against differential power analysis. *Lecture Notes in Computer Science*, 1965:231–??, 2001.

[19] Jordi Cortadella, Alex Kondratyev, Luciano Lavagno, and Christos P. Sotiriou. Desynchronization: Synthesis of asynchronous circuits from synchronous specifications. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(10):1904–1921, 2006.

[20] Crescenzo D'Alessandro, Delong Shang, Alexandre V. Bystrov, Alexandre Yakovlev, and Oleg V. Maevsky. Multiple-rail phase-encoding for NoC. In *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 107–116, 2006.

[21] I. David, R. Ginosar, and M. Yoeli. An efficient implementation of boolean functions as self-timed circuits. *IEEE Transactions on Computers*, 41:2–11, 1992.

[22] D. A. Edwards and A. Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, 45 (1):12–18, jan 2002.

[23] H. Eves. *Elementary Matrix Theory*. Dover publications, 1980.

[24] B.J. Falkowski and S. Rahardja. Efficient computation of quaternary fixed polarity Reed-Muller expansions. *Computers and Digital Techniques, IEE Proc.*, 142:345–352, 1995.

[25] Bogdan J. Falkowski and Cicilia C. Lozano. Quaternary fixed-polarity Reed-Muller expansion computation through operations on disjoint cubes and its comparison with other methods. *Computers & Electrical Engineering*, 31:112–131, 2005.

[26] Karine Gandolfi, D. Naccache, C. Paar, Karine G, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results, 2001.

[27] Minxi Gao, Jie-Hong Jiang, Yunjian Jiang, Yinghua Li, Subarna Sinha, and Robert Brayton. MVSIS. In *Notes of the International Workshop on Logic Synthesis*, June 2001.

[28] D.H. Green. Reed-Muller expansions with fixed and mixed polarities over GF(4). In *IEE Proc., Part E*, volume 137, 1990.

[29] D.H. Green and I.S. Taylor. Multiple-valued switching circuit design by means of generalized Reed-Muller expansions. *Digital Processes*, 2:63–81, 1976.

[30] Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics*. Addison-Wesley, 4th edition, 1999.

[31] Sylvain Guilley, Philippe Hoogvorst, Yves Mathieu, Renault Pacalet, and Jean Provost. Cmos structures suitable for secured hardware. In *In Proc. Design, Automation and Test in Europe (DATE '04)*, 2004.

[32] H.M. Heys. A tutorial on linear and differential cryptanalysis. Technical Report CORR 2001-17, University of Waterloo, March 2001.

[33] Intrinsity, Inc. *Technology White Papers*, chapter 8: N-ary Circuits: Robust Gate Design. www.intrinsity.com, 2006.

[34] A. Jabir, D. Pradhan, and J. Mathew. Gfxpress: A technique for synthesis and optimization of gf(2^m) polynomials. *IEEE Trans. CAD*, 27(4):690–711, April 2008.

[35] A.M. Jabir and D.K. Pradhan. A graph-based unified technique for computing and representing coefficients over finite fields. *Computers, IEEE Transactions on*, 56(8):1119 –1132, aug. 2007.

[36] Dragan Jankovic, Radomir S. Stankovic, and Claudio Moraga. Optimization of GF(4) expressions using the extended dual polarity property. In *Proc. of ISMVL '03*, page 50. IEEE Comp. Soc., 2003.

[37] Cheoljoo Jeong and Steven Nowick. Block-level relaxation for timing-robust asynchronous circuits based on eager evaluation. In *Proceedings of the 14th International Symposium on Asynchronous Circuits and Systems (ASYNC'08)*, 2008.

[38] Mikael Kerttu, Per Lindgren, Mitchell A. Thornton, and Rolf Drechsler. Switching activity estimation of finite state machines for low power synthesis. In *ISCAS (4)'02*, pages 65–68, 2002.

[39] P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks, 1998.

[40] P. Kocher, J. Jaffe, and B Jun. Differential power analysis. In *Proc. CRYPTO*, pages 388–397, 1999.

[41] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. pages 104–113. Springer-Verlag, 1996.

[42] Yuji Kukimoto. BLIF-MV. Technical report, The VIS Group, University of California, Berkeley, 1996.

[43] Konrad J. Kulikowski, Vyas Venkataraman, Zhen Wang, and Alexander Taubin. Power balanced gates insensitive to routing capacitance mismatch. In *Proc. the Design, Automation and Test in Europe Conference and Exhibition (DATE'08)*, pages 1280–1285, 2008.

[44] Mitsuru Matsui. Block encryption MISTY. Communications Science and Techniques, ISEC96-11, 1996.

[45] Peggy B. McGee, Melinda Y. Agyekum, Moustafa A. Mohamed, and Steven M. Novick. A level-encoded transition signaling protocol for high-throughput asynchronous global communication. In *Proc. ASYNC'08*, 2008.

[46] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor. Improving smart card security using self-timed circuits. *Proc. of Asynchronous Circuits and Systems*, pages 211–218, 2002.

[47] Siva G. Narendra and Anantha Chandrakasan. Taxonomy of leakage: Sources, impact, and solutions. In Anantha P. Chandrakasan, editor, *Leakage in Nanometer CMOS Technologies*, Integrated Circuits and Systems, pages 1–19. Springer US, 2006.

[48] J. Pan, J.I. den Hartog, and J. Lu. You cannot hide behind the mask: Power analysis on a provably secure s-box implementation. In *Proc. the 10th International Workshop on Information Security Applications (WISA2009)*, 2009.

[49] I. Poliakov, V. Khomenko, and A. Yakovlev. Workcraft — a framework for interpreted graph models. In *PETRI NETS'09: Proceedings of the 30th International Conference on Applications and Theory of Petri Nets*, pages 333–342, Berlin, Heidelberg, 2009. Springer-Verlag.

[50] Ivan Poliakov. *Interpreted Graph Models*. PhD thesis, Newcastle University, May 2011.

[51] Ivan Poliakov, Danil Sokolov, and Andrey Mokhov. Workcraft: A static data flow structure editing, visualisation and analysis tool. In *ICATPN*, pages 505–514, 2007.

[52] D.K. Pradhan. A theory of galois switching functions. *IEEE Transactions on Computers*, 27(3):239–248, Mar. 1978.

[53] Ashur Rafiev, Andrey Mokhov, Frank P. Burns, Julian P. Murphy, Albert Koelmans, and Alex Yakovlev. Mixed radix Reed-Muller expansions. *IEEE Trans. Comp.*, 2011. Accepted for publication.

[54] Ashur Rafiev, Julian Murphy, Danil Sokolov, and Alex Yakovlev. Bitwise gate grouping algorithm for mixed radix conversion. In *20th UK Asynchronous Forum*, 2008.

[55] Ashur Rafiev, Julian Murphy, Danil Sokolov, and Alex Yakovlev. Conversion driven design of binary to mixed radix circuits. In *Proc. ICCD*, 2008.

[56] Ashur Rafiev, Julian Murphy, and Alex Yakovlev. RTL implementations of GF(2) and GF(4) arithmetic components. Technical report, Newcastle University, 2008.

[57] Ashur Rafiev, Julian Murphy, and Alex Yakovlev. Secure design flow for asynchronous multi-valued logic circuits. In *Proc. to International Symposium on Multi-Valued Logic, ISMVL 2010*, May 2010.

[58] Ashur Rafiev, Julian P. Murphy, and Alex Yakovlev. Quaternary Reed-Muller expansions of mixed radix arguments in cryptographic circuits. In *Proc. 39th International Symposium on MVL*, 2009.

[59] S. Rahardja and B.J. Falkowski. Efficient algorithm to calculate Reed-Muller expansions over GF(4). *Circuits, Devices and Systems, IEE Proc.*, 148:289–295, 2001.

[60] M. Renaudin and F. Bouesse. High security smartcards. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE'04)*, 2004.

[61] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. Cryptographic communications system and method. US Patent No. 4405829, Sep 1983.

[62] Joseph Rotman. *Galois Theory*. Springer-Verlag, second edition, 1998.

[63] RSA Laboratories. *PKCS #1: RSA Cryptography Standard*.

[64] T. Sasao. Input variable assignment and output phase optimization of pla's. *Computers, IEEE Transactions on*, C-33(10):879 –894, oct. 1984.

[65] C.L. Seitz. System timing. In C.A. Mead and L.A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[66] Victor P. Snaith. *Groups, Rings and Galois Theory*. World Scientific Publishing, 1998.

[67] Danil Sokolov, Julian Murphy, Alexander Bystrov, and Alex Yakovlev. Design and analysis of dual-rail circuits for security applications. *IEEE Trans. Comput.*, 54(4):449–460, 2005.

[68] Jens Sparsø and Steve Furber, editors. *Principles of asynchronous circuit design - A systems Perspective*. Kluwer Academic Publishers, December 2001.

[69] John Teifel. Asynchronous cryptographic hardware design. In *Proc. to Carnahan Conferences Security Technology*, 2006.

[70] Kris Tiri and Ingrid Verbauwhede. A vlsi design flow for secure side-channel attack resistant ics. *Design, Automation and Test in Europe Conference and Exhibition*, 3:58–63, 2005.

[71] W. B. Toms, D. A. Edwards, and A. Bardsley. Synthesising heterogeneously encoded systems. In *Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC '06)*, 2006.

[72] E. Tuncer, J. Cortadella, and L. Lavagno. Enabling adaptability through elastic clocks. In *Proc. 46th Design Automation Conference (DAC)*, July 2009.

[73] Newcastle University. Cryptographic processing and processors. UK Patent No. 0719455.8, Oct 2008.

[74] P. Wayner. Code breaker cracks smart cards? *New York Times*, page D1, 22 Jun 1998.

[75] X. Wu, X. Chen, and S.L. Hurst. Mapping of Reed-Muller coefficients and the minimisation of exclusive OR-switching functions. In *Proc. Computers and Digital Techniques*, 1982.

[76] A. Yakovlev, F. Burns, A. Bystrov, and A. Koelmans. Security evaluation of balanced 1-of-n circuits. *IEEE Transactions on VLSI Systems*, 19:2135–2139, 2011.

[77] S. Yanushkevich, D. Popel, V. Shmerko, V. Cheushev, and R. Stankovic. Information theoretic approach to minimization of polynomial expressions over GF(4). In *Proc. of ISMVL '00*, page 265, 2000.

[78] G. Yee and C. Sechen. Dynamic logic synthesis. In *Proc. CICC 1997*, 1997.

[79] Y.Z. Zhang and P.J. W. Rayner. Minimisation of Reed-Muller polynomials with fixed polarity. *IEE Proc.*, 131:177–186, 1984.

[80] Yu Zhou, Danil Sokolov, and Alex Yakovlev. Cost-aware synthesis of asynchronous circuits based on partial acknowledgement. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 158–163, 2006.

[81] Zeljko Zilic and Zvonko Vranesic. Current-mode CMOS Galois field circuits. In *Proc. 23rd International Symp. on MVL*, pages 245–250, 1993.

Index