A VLSI ARCHITECTURE AND THE FPGA IMPLEMENTATION FOR MULTI-RATE LDPC DECODING

A VLSI Architecture and the FPGA Implementation for Multi-Rate LDPC Decoding

By

Mark Jobes, B.A.Sc. (Electrical Engineering)

May 2009

A Thesis

Submitted to the Department of Electrical and Computer Engineering

and the Committee on Graduate Studies

of McMaster University

in partial fulfillment of the requirements

for the degree of

Master of Applied Science

MASTER OF APPLIED SCIENCE (2009) McMaster University

(Electrical and Computer Engineering) Hamilton, Ontario

TITLE:	A VLSI ARCHITECTURE AND THE FPGA IMPLEMENTATION FOR MULTI-RATE LDPC DECODING
AUTHOR:	Mark Jobes, B.A.Sc. (Electrical Engineering)
SUPERVISOR:	Dr. Nicola Nicolici
NUMBER OF PAGES:	xiii, 114

Abstract

Low density parity check (LDPC) codes are utilized frequently in practice as a means of forward error control or error detection. This thesis intends to provide a very large scale integration (VLSI) architecture and corresponding field programmable gate array (FPGA) based implementation for a multi-rate LDPC code.

Utilizing inherent properties in the quasi-cyclic parity check matrix construction for multiple rates, a nested node situation is exploited. This exploitation produces an architecture that, for this thesis' case study of 802.15-3c, has area savings in the check node update calculation block of roughly half the design that would support multiple rates via parallel instantiation of nodes inside the check node update calculation block.

Acknowledgements

Herein I intend to acknowledge most of the individuals and groups whom have aided in the development of this work. Firstly I would like to thank the administrative staff in the ECE department at McMaster University; they greatly facilitated the process entailed in developing this work.

Next I would like to acknowledge my supervisor Dr. N. Nicolici for his great patience, guiding wisdom, and no-nonsense approach to life; this work would have not been at all possible without his help and knowledge.

I would also like to acknowledge Dr. S. Hranilovic and Dr. J.K. Zhang for their aid in the theoretical aspects of this work; without their clarifications, this work would have not been possible in the given timeframe.

I would also like to acknowledge my mates in the CADT group for their endurance of my odd behaviour and for their support. Henry Ko, Adam Kinsman, Ehab Anis, Zahra Lak, Kaveh Elizeh, Jason Thong, Roomi Sahi, and Phil Kinsman; thank you.

Finally I would like to acknowledge Dr. N. Nicolici, Dr. S. Shirani, and Dr. J.K. Zhang for acting as the examination committee for review of this thesis; thank you for your efforts.

Glossary

Abstraction
- The removal of detail to facilitate design description

Architecture
- Graphical or text based schematic of a design in the sense of the circuit

Automation
- The handing over of control to computers for management of design details

CAD, computer aided design,
- Tools for design automation

Check Node
- Container for extrinsic information outside variable nodes

Design space
- The abstract space wherein possible design criteria decisions produce displacement

Design reuse
- The use of already designed cores in a system to ease the design process

Design verification
- The comparison of design against specification

Diagnosis
- Determining what caused a failed device to fail

Decode
- The process of retrieving an information vector from an encoded vector

Encode
- The process of applying an error correcting code to an information vector

FPGA, field programmable gate array
- An ASIC device which may be programmed, via internal SRAM cells, as to simulate the logic network described by the programmed bit stream

HDL, hardware description language
- Design language for RTL descriptions e.g. Verilog/VHDL

IC, integrated circuit
- Many transistors integrated in a single package, a device

Irregular Code
- A code wherein the degree of check nodes varies with that of the variable nodes

Layout
- Low level description of a circuit in terms of transistor placements

LDPC, Low Density Parity Check
- An linear block error correcting code wherein the parity check matrix is sparse

LUT, look up table
- A device with address dependant outputs which can be programmed

Modularity
- Ease of insertion and connection of a module into a system architecture

Moore's law
- An empirical observation of exponential transistor density growth

Netlist
- A low-mid level description of a circuit in terms of logic gate connections

Parity Check Matrix
- A matrix used in linear block error correction

Physical layout see layout

Rate
- One minus the ratio of check nodes to variable nodes

Regular Code
- A code wherein the degree of check nodes is constant along with that of the variable nodes

RTL, register transfer level
- A mid-high level description using register operations

SRAM, Static Random Access Memory
- A type of semiconductor memory that need not be refreshed periodically, yet content values are still lost in the event of power loss

Scalability
- The measurement of how design complexity changes with input problem size

Specification
- The formal description of the design requirements for a system

Synthesis
- Transformation of a description to a lower level of abstraction

Synthesis (architectural)
- Behavioural expressions produce RTL

Synthesis (logic)
- Logic equations produce netlist

Testbench
- Stimuli and expected responses used by simulation during verification

Variable Node
- Container for extrinsic information outside check nodes

VLSI, very large scale integrated circuits
- Containing millions of transistors

Contents

List of Tables

List of Equations

List of Figures

Chapter 1

1 - Introduction

Current technology has achieved a level of complexity that would astound almost any individual from the past. We currently have devices able to perform in mere seconds what would previously take on the order of years. None of this would be possible without integrated circuits (ICs) which are solid state devices created in semiconducting material such as arsenic, germanium, and silicon. Such ICs have seemingly endless application due to inherent generality; i.e. almost any equation or algorithm can be equated or modeled using the power of ICs. ICs are comprised mostly of transistors but some passive components such as resistors and capacitors have proven useful in some applications. In most practical devices today the complexity of the ICs utilized is phenomenal and this complexity is only increasing according to [5]. This increasing tendency could be due to Moore's Law, a long-term trend of computer hardware where the number of transistors inexpensively place-able on a chip increases exponentially as time progresses [5]; a figure graphing this is provided below (re-coloured from [32]) in Figure 1.1.

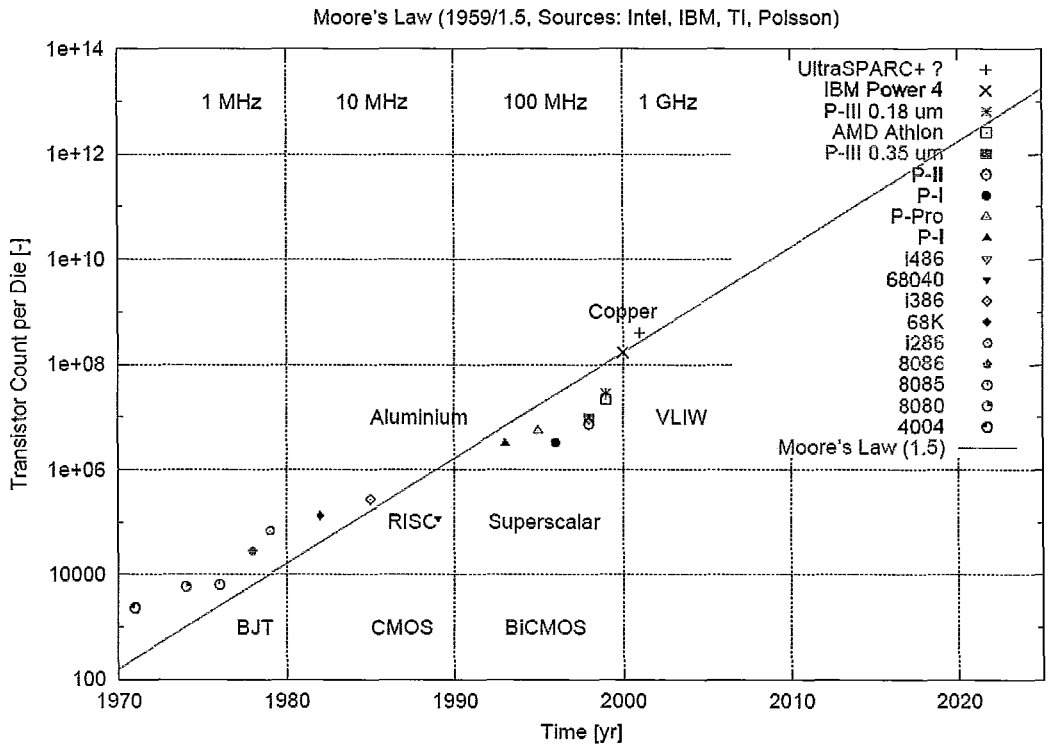


Figure 1.1: A graph of transistor count vs. time from [32].

In the following sections of this chapter, discussion will be provided on the general design of Very Large Scale Integration (VLSI) circuits (Chapter 1.1), along with the modularity and abstraction utilized (Chapter 1.1.2), and the automation of design steps with use of computer aided design (CAD) tools (Chapter 1.1.3). Also discussion will be provided on low density parity check (LDPC) codes (Chapter 1.2), the basics of LDPCs (Chapter 1.2.1), the process of encoding (Chapter 1.2.2), the process of decoding (Chapter 1.2.3), and the process of parity check matrix construction (Chapter 1.2.4).

1.1 - VLSI Design

To directly design some of today's complex devices at the lower levels are complex. The transistor and fabrication levels require a vast amount of criteria to be established and resolved; this could take a single individual as long as his or her own life span. Thus, to be able to produce the more complex designs of today, some form of automation along with a consistent design methodology is a necessity. To create an automation process that could directly take high level expressions and immediately translate these expressions into fabricated ICs is a task that is as daunting, if not more so, when compared to the original problem of layout or transistor level design; another solution is needed.

1.1.1 - Design Modularity and Abstraction

Modular design allows automation to be used to create solutions automatically from high level expressions is feasible. By utilizing levels of abstraction, problems can be broken down into much more manageable sub-problems and then solutions can be linked together.

Abstraction can be thought of as the process of intelligent sectioning of stages or modules or simply information as to produce a generalized viewpoint or

module or section, in which only the immediately relevant information is incorporated. For example when organizing a company from the highest level, one may use levels of abstraction to simplify the payroll and then need only be concerned with section payroll; leaving the details in each section to be dealt with by the section supervisor(s).

By chaining different tools, or in other words cascading solutions, across different levels of abstraction, a design may be produced in a realistic timeframe, i.e. within the time-to-market. Another important point is that if a sub-problem is solved, yet needed multiple times, there is a significant reduction in the total amount of work needed due to the ability of module reuse. This is one of the most significant features of modular design. Due to the nature of modular design a sub-problem need only be solved once regardless of the number of times the sub-problem is needed. This is because the solution can simply be copied and then it may be linked into the final solution.

The means for the sub-problem solving and later linking of solutions is encapsulated in the field of design automation. Computers may be used in tandem with powerful algorithms as to optimise and compile higher level expressions or languages into the desired lower level expressions or languages; eventually reaching the lowest level of design. This is the subject of the following section.

1.1.2 - Automation and CAD Tools

Modular design may seem to be a sufficient solution to the problem of determining solutions to large scale problems yet is insufficient without the aid of automation. Truly a sub-problem could be solved and then could be linked into the larger solution by hand, but since this process is strongly deterministic (i.e. the input and output problems are bounded in addition to a unique input consistently resulting in the same corresponding output), automation is possible. Automation is the solution for when the same set of steps need to be performed multiple times; this occurs frequently in the scope of IC design, such as when functions are optimized or netlists processed. Automation also has the advantage of modularity. For instance an automatic tool need not be holistic; the tool may only solve part of a sub-problem. This result could then be linked with the other automation tools necessary for a complete solution to the sub-problem at hand.

Even with all the CAD tools at hand, a means of cascading or linking the automatically generated data is still required as to produce a design; to perform this cascading or linking, an order or dependency first need be established. In the following section a description of the CAD based implementation technology flow (i.e. design order) is provided.

1.1.3 - Implementation Technology

In this section of the thesis the flow through the various stages of design expression, as understood by this thesis, to final implementation is provided.

The various stages of design flow adopted by this thesis are resultant of the levels of abstraction utilized and the CAD tools available. These levels include: behavioural modeling, logic synthesis, technology mapping, and simulation and verification. A figure is provided below as to illustrate this flow, Figure 1.2, followed by a delineation of the illustrated stages.

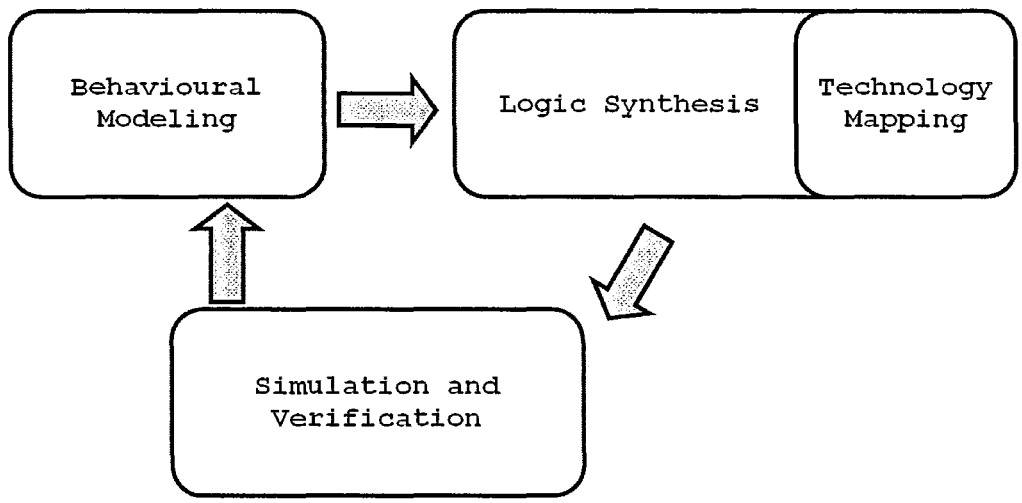


Figure 1.2: Design flow adopted by this thesis.

Behavioural modeling, similar to algorithmic synthesis or high-level synthesis is an automated design process that interprets an algorithmic

description and produces the corresponding hardware [30]. The algorithmic description is of a desired behaviour and the hardware created implements said behaviour. Initially ANSI C/C++/SystemC code or similar is produced as to describe the desired behaviour, the code is then analyzed, architecturally constrained, and scheduled. This produces register transfer level (RTL) description in a hardware design language (HDL) such as Verilog [45]. This HDL, with the use of a logic synthesis tool, may then be synthesized to the gate level for technology mapping.

Logic synthesis is the automated design process that interprets a description of design in terms of logic specification, such as RTL, and produces a structural view of an equivalent logic-level model [30]. An example of this is the synthesis of Verilog [45] HDL into a gate-level netlist; a circuit described as a netlist wherein the basic building blocks described by the implemented library are connected as to produce the desired behaviour [30]. This stage of library binding also known as technology mapping is considered the backend of the logic synthesis stage [30] and is further outlined in the following.

Technology mapping is the automated design process that interprets an unbound logic network and then binds the network, utilizing a cell-library, to a gate-level netlist [30]. An example can be constructed when regarding a 3-input AND gate. The AND between two inputs is the binary operation in which the

following must be true: AND(0,0)=0, AND(0,1)=AND(1,0)=0, and AND(1,1)=1. The AND between three inputs may be thought of as AND(AND($input_1$,$input_2$),$input_3$) or as AND($input_1$,$input_2$,$input_3$). This is indicative of the nuances between an unbound logic network and a post-mapped bound logic network. If the unbound equation of a 3-input AND were to be mapped/bound using a library in which 3-input ANDs did not exist, the equation would become AND(AND($input_1$,$input_2$),$input_3$). Alternatively if 3-input ANDs did exist in the library then the equation would become AND($input_1$,$input_2$,$input_3$). In short the stage of technology mapping allows the unbound logic network, provided by the frontend of the logic synthesis stage, to be mapped to a bound logic network which utilizes the standard building blocks provided in the implemented library [30]. An example of a technology to which a design may be bound is that of a Field Programmable Gate Array (FPGA). FPGAs are devices that are capable of implementing these logic networks using look-up-tables (LUTs) as a standard building block in addition to some device specific blocks such as multipliers, digital signal processing (DSP) blocks, and memory blocks. LUTs are devices that can have logic functions programmed as functions of the LUT input; simply LUTs are analogous to truth tables and are discussed in more detail in section 2.2. An FPGA is illustrated below in Figure 1.3. Note how the interconnection of LUTs and other internal blocks can be configured as to have

almost any design of compatible complexity. These LUTs are built from static random access memory (SRAM) cells in FPGAs from Altera and Xilinx, yet logic may be implemented with different standard cells such as NAND gates. It should be noted that the number of cells in a FPGA is finite.

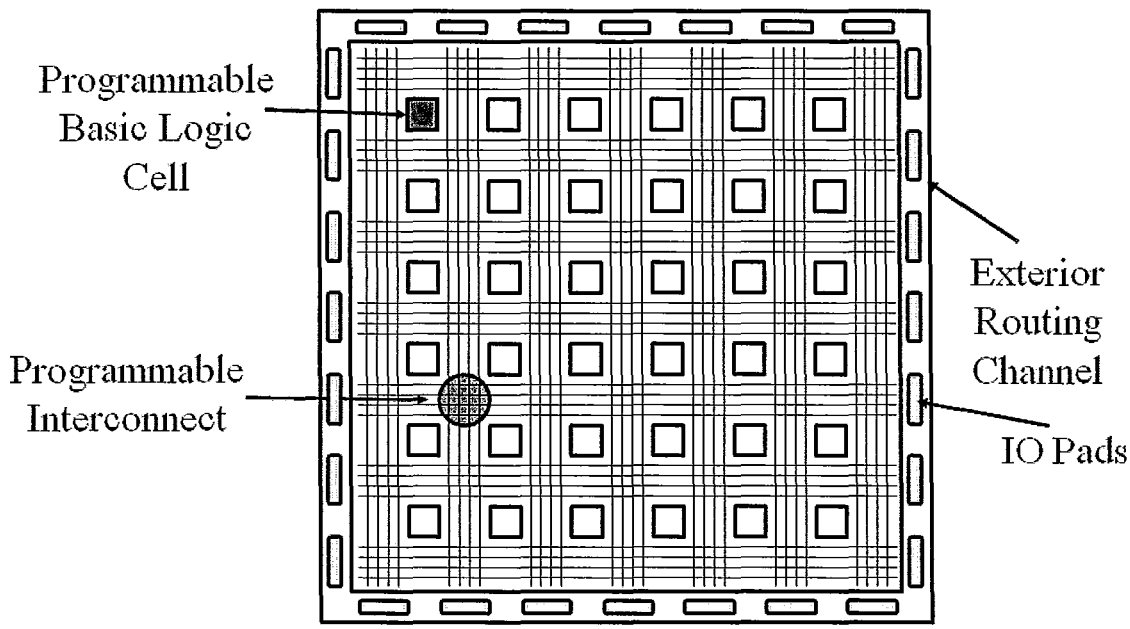


Figure 1.3: Symbolic representation of an FPGA.

Once the design has been created or even once modules of a design are created, it or they may then be simulated as to determine correct functionality. The purpose of this is to avoid the situation wherein the final design is implemented and nothing in the design, or only some of the design, functions correctly. Proper simulation and verification (see Figure 1.2) of a design is a

necessity; without proper simulation and verification, errors would be near impossible to locate. Many simulation tools exist; from timing simulators to event driven simulators to combinations of both. Once a design is verified the process to produce in silicon may commence. It should be noted that merely stating 'produced in silicon' is not representative of the great effort required in this stage. This is discussed in the following.

Once a design is verified to have the correct functionality the scope of the validation test needs to be recognized; i.e. for a design to function on an FPGA far from guarantees the successful implementation of the design in an application-specific integrated circuit (ASIC). This is obvious when noting the differences amongst the standard cells (i.e. logic gates vs. LUTs) along with the additional complexity involved in ASIC implementation (i.e. layout, routing, layers, and masks). The details of the ASIC design steps are shown below in Figure 1.4; followed by an elaboration of the encompassed steps.

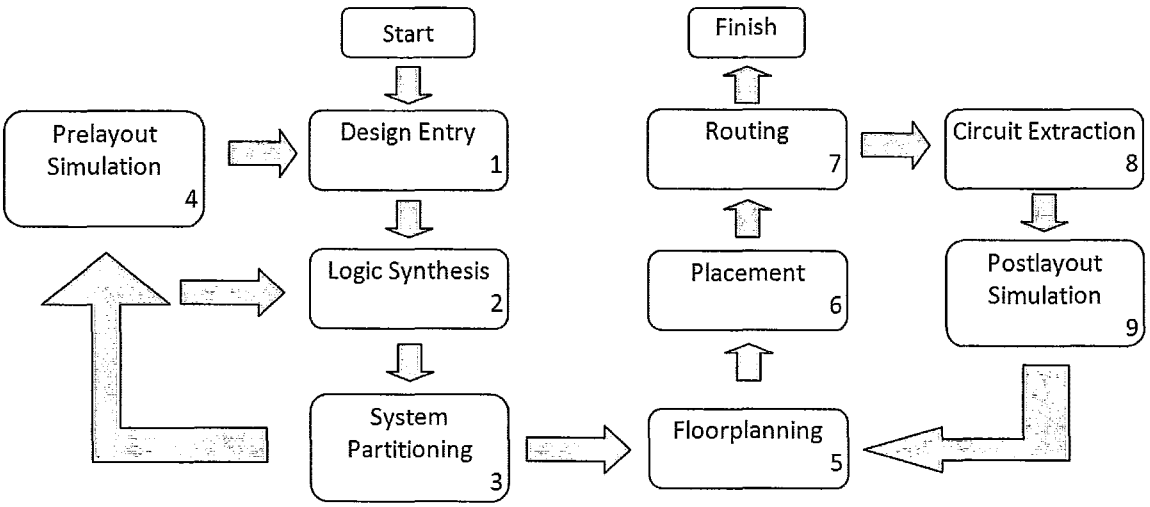


Figure 1.4: ASIC design flow redrawn from [42].

In the design entry stage, the circuit to be designed is provided to the flow as either HDL or simply a schematic. The logic synthesis stage is consistent with its earlier explanation and as a result produces the gate level netlist. In system partitioning the large system is divided into ASIC-sized pieces [42]. Prelayout simulation verifies design functionality and since the design is virtually represented, hence pre-layout, this stage is far from a formal verification of the final implementation. Floorplanning arranges the blocks in the netlist onto the layout of the chip. Placement arranges the cells inside each of the blocks. Routing interconnects the cells and blocks with wires. Circuit extraction is the stage wherein the resistive properties of the wires, whose lengths and locations are resultant from the routing stage, in addition to the capacitive properties, are

calculated. Finally postlayout simulation is performed as to determine if the circuit will still function now that the load parameters determined by the extraction can be incorporated into the calculations and analysis.

The ASIC design steps earlier elaborated can be utilized as to produce any digital circuit. In this thesis, these design steps will not be used for an ASIC targeted device as an ASIC device is not what was intended to be created; i.e. a chip was not fabricated. Instead the earlier design flow from Figure 1.2 will be used to create an FPGA implementation of a LDPC decoder with multi-rate support.

In the following section low density parity check codes will be described with: the basics in section 1.2.1, the encoding process in section 1.2.2, the decoding process in section 1.2.3, and the parity check matrix construction in section 1.2.4.

1.2 - LDPC Codes

In the digital world of circuits a single flaw, or bit flip, can cause catastrophic errors [23]. In an attempt to reduce this loss of information from occurring, error correcting codes have been developed [23]. The principal concept is that redundancy is added into the transmitted word as to provide the needed information for correcting the incorrect bits in the transmitted word (or simply to detect errors). There are a myriad of means to implement redundancy ranging from a simple parity check bit (i.e. a single bit that is an XOR function of other bits in the same word) to a matrix based, or linear block, method. One type of linear block based error correcting code is the low density parity check code (LDPC code). Developed by Gallager in the 1960's [11], LDPC codes received little attention until the invention of Turbo Codes which utilized a similar decoding concept based on an iterative algorithm [2]. In 1996 Mackay and Neal rediscovered LDPC codes and realized the significant error correcting capabilities that LDPC codes could yield [2]. Not only could LDPC codes correct errors but they were also capacity-approaching codes [10], [11], [15]. Capacity-approaching codes are those codes whose transmission rates can approach (but never achieve due to the limitations of the system, i.e. power is attenuated, energy conserved) the Shannon limit [39] (a theoretical limit devised by Shannon and

Hartley) over a channel. Channels such as: a binary symmetric channel, binary erasure channel, or an additive Gaussian White Noise channel [39].

1.2.1 - LDPC Coding Basics

LDPC codes are a subset of the family of codes encompassing all linear block codes [7] wherein said subset differs in the sense that the corresponding parity check matrix is sparse, hence the low density, and normally large. The code is large or simply long, as to achieve the good performance associated with LDPC codes [39]. LDPC codes are represented either by a Tanner graph (as in [44]) or simply, and more frequently, by the parity check matrix. The relation of the Tanner graph (a bipartite graph) to the parity check matrix is described pictorially, for the general case, below in Figure 1.5.

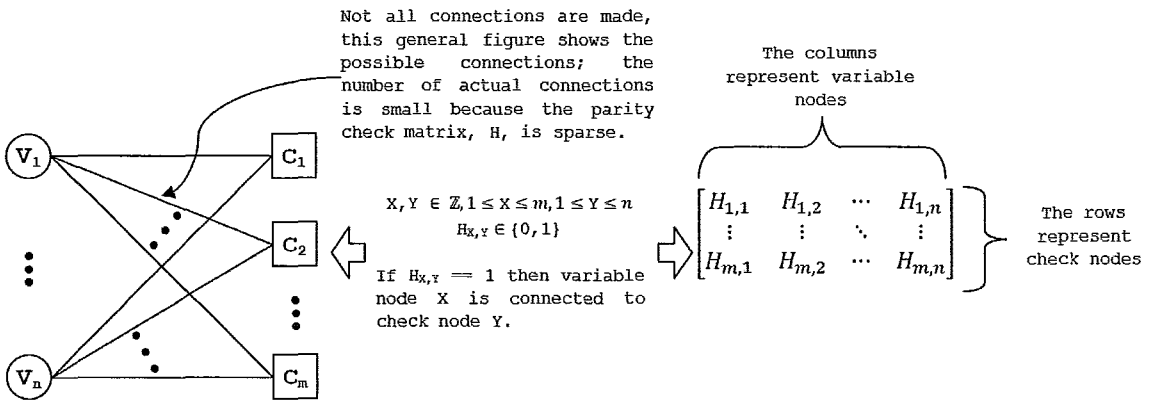


Figure 1.5: A general representation of the Tanner graph (left) and parity check matrix (right).

Regarding the general Tanner graph in Figure 1.5, the circles represent the variable nodes (one for each of the bits in the transmitted word, including redundancy), and the squares represent the check nodes (count depends on code rate). The concept of nodes may be vague, if so allow variable nodes to be information storage containers that hold information about the transmitted word and check nodes to be information storage containers that hold information about how the transmitted word relates to itself (i.e. how errors are found and potentially corrected). The matrix to the right of the Tanner graph is the matrix representation of the same code, its contents either a 1 meaning a connection or a 0 meaning no connection.

The throughput and design complexity of LDPC codes are a result of only a few design parameters including: regular vs. irregular codes, the code block length, code rate, the update formulae complexity and accuracy, the interconnection density between nodes, the implementable parallelism, and the number of iterations of the update formulae. These design parameters will be outlined in the following.

If every check node in the LDPC code has the same degree, or number of connected variable nodes, and every variable node has the same number of check nodes connected, the code is considered regular. There are obvious reasons why a regular code would be used, i.e. the consistency in nodal design

yields a simpler hardware implementation. However in low signal to noise (SNR) regimes irregular codes [27] have a better performance [3], [18], and [20]. The higher degree nodes converge faster and then help the lower degree nodes to converge [18]. Because of this improved convergence, a reduction in total area relative to performance is experienced when selecting an irregular structure as opposed to a regular structure. Thusly irregular codes are used more frequently in industry.

The code block length is the number of bits in the transmitted word (including the added redundancy); i.e. the number of variable nodes. As the code block length increases, the LDPC code approaches capacity [3]. The exact reason why such occurs is out of the scope of this thesis and is addressed in [11]. However, a simple reason may be portrayed when regarding that, with a constant overhead or number of redundant bits, there is a reduction in the relative overhead in the transmitted word as the code block length increases.

The code rate is the ratio of the number of information, or original (pre-redundancy), bits over that of the code block length. For example with a code rate of one half, the number of bits added for redundancy is equal to that of the number of information bits. Alternatively, if the code rate is one third then there are twice as many bits for redundancy as there are for original information. As the code rate increases, recall this is regarding an irregular LDPC code, irregularity

decreases [3] yet error correction capability also decreases. The reason why the irregularity decreases when increasing the rate is because the number of check nodes in comparison to variable nodes decreases, making it more difficult to maintain a sparse graph which is a necessity for successful decoding [3].

The update formulae choice and accuracy have a drastic impact on design performance and complexity. The formulae are used to update the values of the messages passed between nodes. This is known as a belief propagation algorithm and is addressed in detail in section 1.2.3. If these messages require complex calculations to be updated it is obvious why this increases the complexity of the circuit, especially when recalling that the calculations have to be done for each node in the code. The impact of the chosen accuracy, or quantization, of the code is manifested in the update calculation. Even if the update calculation utilizes the least expensive solution attainable, if the quantization is larger than necessary to guarantee convergence of the belief propagation algorithm, a significant portion of area is wasted. Therefore, a quantization should be chosen in tandem with the update formulae as to achieve the most resource efficient configuration, for the code in question, that achieves the desired performance.

The interconnection density between the set of variable nodes and check nodes has an obvious relation to design complexity when regarding that more

connections essentially entail more wires to route and more values to update and calculate. As density increases, the degree of each of the nodes will increase thus increasing the amount of information used for the calculation; this improves the convergence but also increases design complexity. Also as the density of the code increases, the sparseness of the code decreases thus degrading the performance [3].

Regarding implementable parallelism, the commonly accepted concepts that: increasing parallelism decreases computational time and increases area (i.e. size of the digital circuit, measured with a standard blocks or even at the transistor level); and that decreasing parallelism increases computational time and decreases area, both apply. The main point of concern is how easily the parallelism may be controlled. If a code has been designed to be hardware efficient parallelism is facilitated [36]. The codes that seem to be most facilitating are those of the quasi-cyclic class [6], the reason is because in addition to any common parallelism (at the algorithmic level) the parity check matrix may be sectioned into blocks and each block may be processed independently. Practical applications such as: DVB-S2, WLAN (802.11n), and WiMAX (802.16e) utilize quasi-cyclic or augmentations of quasi-cyclic codes see [1], [6], [8], [24], and [33].

The number of iterations that are needed is inversely proportional to the throughput and is thusly a significant parameter. With a particular code and

algorithm, if the number of allowable iterations is decreased as to increase throughput, the quantization must be increased to maintain a comparable performance [3]. Maintaining the number of iterations to a minimum is desired. To facilitate this need to minimize iteration count there exist different decoding schemes such as the layered method where iteration count can be reduced by as much as a factor of two [15]; addressed later in section 1.2.3.

1.2.2 - Encoding

Encoding techniques have changed greatly since the traditional method of utilizing a generator matrix, called G in this thesis. G could be derived almost directly from the parity check matrix, H. Such is done by reducing H into a particular form via row operations and then, through internal reorganizing, G is derived; a clear complete encoding example is below in Figure 1.6. It should be noted that Rx, where x is an integer, is a handle for row x. For example R3 ← R1⊕R3 means take rows 1 and 3, bitwise (bit by bit) XOR them and place the result in row 3.

Given a partiy check matrix H, if row operations are performed as to force H into the form:

$$H' = [-P^T \mid I_{n-m}] \text{ then}$$
$$G = [I_m \mid P],$$

where n is the number of variable nodes and m is the number of check nodes. Note that this is a binary code thus $-P = P$.

$$H = \begin{bmatrix}1&0&1&1&0&0\\0&1&1&1&0&1\\0&1&0&1&1&0\end{bmatrix} \rightarrow R3 \leftarrow R1 \oplus R3 \Rightarrow \begin{bmatrix}1&0&1&1&0&0\\0&1&1&1&0&1\\1&1&1&0&1&0\end{bmatrix} \rightarrow \text{Swap}(R2,R3) \Rightarrow \begin{bmatrix}1&0&1&1&0&0\\1&1&1&0&1&0\\0&1&1&1&0&1\end{bmatrix}$$

$$R3 \leftarrow R1 \oplus R3 \Rightarrow \begin{bmatrix}1&0&1&1&0&0\\1&1&1&0&1&0\\1&1&0&0&0&1\end{bmatrix} = H' \Rightarrow -P^T = \begin{bmatrix}1&0&1\\1&1&1\\1&1&0\end{bmatrix} \Rightarrow P = \begin{bmatrix}1&1&1\\0&1&1\\1&1&0\end{bmatrix} \Rightarrow G = \begin{bmatrix}1&0&0&1&1&1\\0&1&0&0&1&1\\0&0&1&1&1&0\end{bmatrix}$$

To encode x = [1 0 0]

$$\Rightarrow \hat{x} = [1\ 0\ 0]\begin{bmatrix}1&0&0&1&1&1\\0&1&0&0&1&1\\0&0&1&1&1&0\end{bmatrix} = [1\ 0\ 0\ 1\ 1\ 1]$$

To check residue (for the valid codeword):

$$\hat{x}H^T = [1\ 0\ 0\ 1\ 1\ 1]\begin{bmatrix}1&0&0\\0&1&1\\1&1&0\\1&1&1\\0&0&1\\0&1&0\end{bmatrix} = [0\ 0\ 0] \leftarrow \text{Therefore success, zero residue}$$

To check residue (for an invalid word):

$$[1\ 1\ 0\ 1\ 1\ 1]\begin{bmatrix}1&0&0\\0&1&1\\1&1&0\\1&1&1\\0&0&1\\0&1&0\end{bmatrix} = [0\ 1\ 1] \leftarrow \text{Therefore fail, non-zero residue}$$

Figure 1.6: Complete encoding example using the generator matrix derived from the parity check matrix.

Thus regarding the encoding of a linear block code of length N with M check points, through traditional means with a single processing engine, a temporal complexity of O(NM) is required [22], as it is the result of a matrix multiplication of the binary input vector by the generator matrix; note that if M processing engines were utilized that the temporal complexity would be O(N).

Modern encoding is rather different, wherein the parity check matrix H is utilized without a generator; such can be constructed to result in linear temporal complexity encoding [22], [26]. The method is simple and is illustrated below in Figure 1.7.

$$\hat{x}H^T = [\hat{x}_1 \quad \cdots \quad \hat{x}_n]\begin{bmatrix} H_{1,1} & \cdots & H_{m,1} \\ \vdots & \ddots & \vdots \\ H_{1,n} & \cdots & H_{m,n} \end{bmatrix} = [0 \quad \cdots \quad 0] \Leftrightarrow \begin{Bmatrix} [\hat{x}_1 \quad \cdots \quad \hat{x}_n] \cdot [H_{1,1} \quad \cdots \quad H_{1,n}] = 0 \\ \vdots \\ [\hat{x}_1 \quad \cdots \quad \hat{x}_n] \cdot [H_{m,1} \quad \cdots \quad H_{m,n}] = 0 \end{Bmatrix}$$

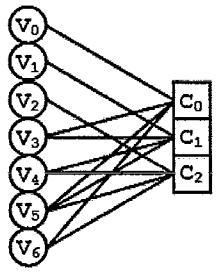
$$\Leftrightarrow \begin{Bmatrix} \hat{x}_1 * H_{1,1} \oplus \ldots \oplus \hat{x}_n * H_{1,n} = 0 \\ \vdots \\ \hat{x}_1 * H_{m,1} \oplus \ldots \oplus \hat{x}_n * H_{m,n} = 0 \end{Bmatrix} \text{ i.e. a system of equations to solve}$$

Figure 1.7: Encoding represented as a system of equations.

In Figure 1.7, the temporal complexity for solving the set of equations may seem to be O(NM) but this can be reduced. According to [22], by designing the parity check matrix to be sparse, the set of equations may be solved immediately without any reduction or elimination algorithms (no order is implied, order depends on the particular construction of H). Therefore, with one processing engine, the temporal complexity may be reduced to O(M), making linear time encoding of an LDPC code feasible; if M processing engines were utilized the complexity can be reduced to constant time. A complete example of the encoding process is shown below in Figure 1.8.

With the below initial conditions:

$$x=[x_0 \quad x_1 \quad x_2 \quad x_3] \qquad H=\begin{bmatrix}1&0&0&0&1&1&1\\0&1&0&1&1&1&0\\0&0&1&1&0&1&1\end{bmatrix}$$



And using the following:

$$\hat{x}H^T=[\hat{x}_0 \quad \hat{x}_1 \quad \hat{x}_2 \quad \hat{x}_3 \quad \hat{x}_4 \quad \hat{x}_5 \quad \hat{x}_6]\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\1&1&0\\0&1&1\\1&1&1\\1&0&1\end{bmatrix}=[0 \quad 0 \quad 0]$$

Allows us to set up a system of equations:

$$\begin{aligned}\hat{x}_0\oplus\hat{x}_3\oplus\hat{x}_5\oplus\hat{x}_6=0 &\Rightarrow \hat{x}_0=\hat{x}_3\oplus\hat{x}_5\oplus\hat{x}_6\\ \hat{x}_1\oplus\hat{x}_3\oplus\hat{x}_4\oplus\hat{x}_5=0 &\Rightarrow \hat{x}_1=\hat{x}_3\oplus\hat{x}_4\oplus\hat{x}_5\\ \hat{x}_2\oplus\hat{x}_4\oplus\hat{x}_5\oplus\hat{x}_6=0 &\Rightarrow \hat{x}_2=\hat{x}_4\oplus\hat{x}_5\oplus\hat{x}_6\end{aligned}$$

With the above eqations we may substitute $\hat{x}_0$, $\hat{x}_1$, and $\hat{x}_2$ back into $\hat{x}$

$$\begin{aligned}\hat{x}&=[\hat{x}_0 \quad \hat{x}_1 \quad \hat{x}_2 \quad \hat{x}_3 \quad \hat{x}_4 \quad \hat{x}_5 \quad \hat{x}_6]\\ &=[\hat{x}_3\oplus\hat{x}_5\oplus\hat{x}_6 \quad \hat{x}_3\oplus\hat{x}_4\oplus\hat{x}_5 \quad \hat{x}_4\oplus\hat{x}_5\oplus\hat{x}_6 \quad \hat{x}_3 \quad \hat{x}_4 \quad \hat{x}_5 \quad \hat{x}_6]\end{aligned}$$

And using the below we may write $\hat{x}$ in terms of x (Encoding Complete)

$$[\hat{x}_3 \quad \hat{x}_4 \quad \hat{x}_5 \quad \hat{x}_6]=[x_0 \quad x_1 \quad x_2 \quad x_3]$$

$$\Rightarrow \hat{x}=[x_0\oplus x_2\oplus x_3 \quad x_0\oplus x_1\oplus x_2 \quad x_1\oplus x_2\oplus x_3 \quad x_0 \quad x_1 \quad x_2 \quad x_3]$$

Figure 1.8: A linear block code encoding example.

In the above figure the information vector is x and is comprised of 4 information bits: x_0, x_1, x_2, and x_3. The parity check matrix, H, is a rate 4/7 code having four information bits and a block length of seven; used to produce and verify the encoded word $\hat{x}$. Because there is an I_{3x3} identity matrix in the beginning of H, the last four bits in the encoded word are copies of the four

original information bits; this is termed systematic and is a property of many implemented LDPC codes such as in the standard DVB-S2 [8], and WiMAX [17]. It is apparent that allowing the result of the multiplication of the coded word with the rotated parity check matrix, termed the residue, to equate to a zero vector greatly facilitates the reordering of equations as to solve the encoding problem. With a sparse parity check matrix, this approach can be computed with linear time complexity [26]; far superior to the method of solving $\hat{x}H^T=0$ as a system of equations via matrix multiplication which could require O(NM) to compute.

1.2.3 - Decoding

Much of the work in the LDPC code design space focuses on the decoding algorithm or heuristic. According to [38], optimal decoding of an LDPC code is an NP-complete problem, there are, however, heuristics based on belief propagation, wherein the context of the application, are considered optimal [36], [37]. Belief propagation is an algorithm that utilizes nodes' extrinsic information (or simply information outside of the node in question) to later make decisions from information that was propagated through the nodal network. The values that are propagated are representations of reliability determined from observing the

communication network channel; these values are represented as log likelihood ratios (LLRs).

The belief propagation algorithm for LDPC decoding has two stages: update the check node extrinsic information, and update the variable node extrinsic information. These two steps may be performed in a single iteration (according to [7] termed: Layered, Turbo or, Gauss-Seidel Decoding) or in two iterations (termed two-phase message passing (TPMP) and is the traditional method according to [17], and [47]). The result is that the number of iterations can change by a factor as much as two [15], [47] when using layered decoding as opposed to TPMP (the chosen scheme for this thesis as to not incur the complexity of layered decoding scheduling). One work deserving specific recognition is that of [47], where a layered decoding scheme is implemented with a novel use of a delay chain built inside the node update calculation blocks.

Of all the decoding algorithms the Sum Product Algorithm (SPA) [21] is determined as the most beneficial with regard to only performance [36]. All of the work cited in this thesis either uses SPA or an augmented flavour of SPA yet all are Belief Propagation Algorithms. The Belief Propagation Algorithms all follow a similar algorithmic flow where there is a formula for updating the messages outputted by the nodes, with an iterative nature, and a formula for producing the binary vector to be tested for success (i.e. the binary vector passes all parity

checks). A general example is given below in Figure 1.9; note how a particular node update is not dependent on itself due to the use of extrinsic information (except for the test binary vector which includes intrinsic information). Extrinsic information is information outside of a particular node, whereas intrinsic information is information inside of a node.

Below are the general forms of all the Belief Propagation Algorithms.

CheckNodeOutput(check node c, to variable node v)= Formula(all variable nodes connected to c ≠v)

VariableNodeOutput(variable node v, to check node c)= Formula(all check nodes connected to v ≠c)

TestVectorOutput(variable node v)= Formula(all check nodes connected to v)

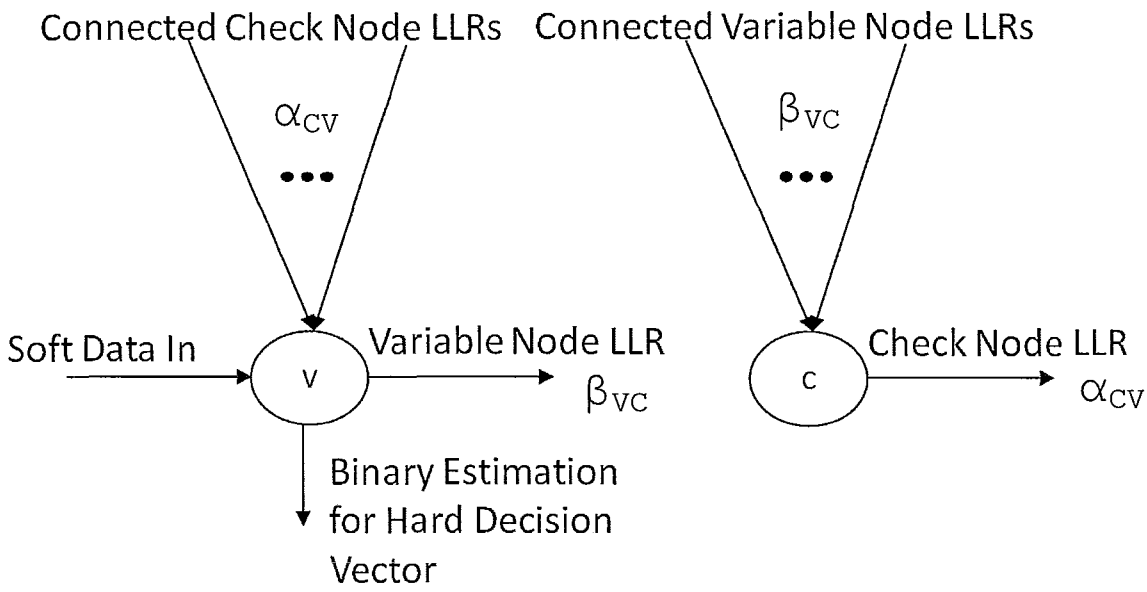


Figure 1.9: General flow of information for update calculation formulae.

In the above figure α represents the 2D vector of check node (subscript C) to variable node (subscript V) information, and β represents the 2D vector of variable node (subscript V) to check node (subscript C) information.

There are three main types of Belief Propagation based decoding algorithms that are commonly utilized: the standard SPA in Equation 1.1 [36], the Min-Sum algorithm (or equivalently the Uniformly Most Powerful algorithm, UMP) in Equation 1.2 [7], and the λ-Min algorithm. The λ-Min algorithm is an augmentation of UMP that realizes the magnitude of the possible results for a particular node at a particular output comprises a set of λ (two commonly, thus either the min or 2nd min); such is used to simplify the hardware design complexity resultant from formulae computation difficulty.

$$\alpha_{cv} = \left(- \prod_{n \in N(c), n \neq v} \operatorname{sgn}(\beta_{nc}) \right) \log \left(\tanh \left| \sum_{n \in N(c), n \neq v} \frac{\log\left(\tanh\left(\frac{\beta_{nc}}{2}\right)\right)}{2} \right| \right)$$

$$\beta_{vc} = \sum_{m \in M(v), m \neq c} \alpha_{mv} - \frac{2Y_v}{\sigma^2} \qquad \beta_v = \sum_{m \in M(v)} \alpha_{mv} - \frac{2Y_v}{\sigma^2}$$

Equation 1.1: SPA Formulae

In the above figure for SPA formulae, β_V is the hard decision vector component for variable node V. N(check node) produces the set of all the variable nodes connected to the check node argument, and M(variable node) produces the set of all check nodes connected to the variable node argument. sgn(x) = {1 for x ≥ 0, -1 otherwise}, or simply the sign of the number argument is equated. When a vector is received, an analogue to digital converter (ADC) expresses each point in the vector in a fixed point manner where values exist in [-1, 1] and these data points comprise the soft data in the context of this thesis. Y_V is the soft data from detection for variable node V and σ is the noise standard deviation of the Additive White Gaussian Noise (AWGN) channel in which the message is transmitted.

$$\alpha_{cv} = \left\{ \bigoplus_{n \in N(c), n \neq v} \text{SignBit}(\beta_{nc}) \, , \, \min|\beta_{nc}| \right\}$$

$$\beta_{vc} = \frac{2Y_v}{\sigma^2} + \sum_{m \in M(v), m \neq c} \alpha_{mv} \qquad \beta_v = \frac{2Y_v}{\sigma^2} + \sum_{m \in M(v)} \alpha_{mv}$$

Equation 1.2: UMP Formulae

In the above figure for UMP, β_v is the hard decision vector component for variable node V. N(check node) produces the set of all the variable nodes connected to the check node argument, and M(variable node) produces the set of all check nodes connected to the variable node argument. SignBit(x) = {1 for x < 0, 0 otherwise} or simply the MSB of a value represented in signed magnitude. The {} brackets are used to represent a concatenation of two binary words; i.e. {a,b} produces the new word ab, or {0,101} produces 0101. The $\oplus$ symbol refers to the XOR or modulo-2 addition operation over all the 1-bit arguments in the following parenthesises. Y_V is the soft data from detection for variable node V and σ is the noise standard deviation of the AWGN channel in which the message is transmitted.

The SPA algorithm does result in a higher area due to algorithmic complexity, yet it also has high performance [3]. The performance relative to area seems insufficient as to justify said algorithm's use for many of the architectures amongst the prior art (see section 2.1 - 2.2); none of the prior art utilizes the more complex formulae without a simplification that results in performance degradation.

1.2.4 - Matrix Construction

The construction method of the parity check matrices is essential for the applicability of this thesis. The construction method must utilize the later described method of expanding a base quasi-cyclic LDPC code with rate of the form (R-1 / R) into the set of matrices to be supported (all with rates of the same form R-1 / R). This construction method can be thought of as the only constraint that this work requires for applicability. When matrices' rates posses equivalent 'form[s]', as is mentioned in this thesis, it is implied that each individual matrix rate is expressible in the stated form (i.e. 4/7 is not of the form R-1 / R, yet 7/8 is of said form). To support alternative rates could possibly be done yet this detail was not explored as the case study had matrices of compatible rate form.

The construction method must utilize a base matrix built from blocks of cyclically permuted matrices, commonly referred to as quasi-cyclic codes [9], [39], [46]. The representation of this is given in a matrix form wherein the values represent the number of cyclic permutations of columns, towards the right, of the identity matrix; the size of the identity matrix is a design parameter, termed K, and is out of the scope of this thesis (please see [23] for details and especially the sections on quasi-cyclic LDPC codes). An example of such a construction method is shown below in Figure 1.10, wherein the cyclic shifts or permutations

are illustrated, and in Figure 1.11 wherein a simple base matrix representation is shown.

$$I_K^0 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \quad I_K^1 = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix} \quad I_K^2 = \begin{bmatrix} 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \end{bmatrix}$$

Figure 1.10: An example of identity matrix permutation for an identity matrix of size K x K.

	Variable Node Block 0	Variable Node Block 1		Variable Node Block m-1
Check Node Block 0	$I_K^{\alpha_{00}}$	$I_K^{\alpha_{01}}$	...	$I_K^{\alpha_{0\,m-1}}$
Check Node Block 1	$I_K^{\alpha_{10}}$	$I_K^{\alpha_{11}}$	...	$I_K^{\alpha_{1\,m-1}}$
	...	...	...	...
Check Node Block n-1	$I_K^{\alpha_{n-1\,0}}$	$I_K^{\alpha_{n-1\,1}}$	...	$I_K^{\alpha_{n-1\,m-1}}$

Figure 1.11: A general example of base matrix contents.

It should be noted that the contents of the base matrix are not necessarily all cyclic permutations of an identity matrix, some of the contents are filled with null matrices (i.e. all zero matrices). Also in Figure 1.11: $\alpha_{ij} \in \mathbb{Z},\ 0 \leq \alpha_{ij} < K,\ i,j \in \mathbb{Z}, 0 \leq i < n, 0 \leq j < m$, or in other words there exist at most n*m entries

of α with K different possible values [0, K-1]; to perform K cyclic permutations on a KxK identity matrix results in the original KxK identity matrix. The general matrix in Figure 1.11 would be for a parity check matrix of size n*k x m*k.

To support multiple rates, more matrices (one for each rate) will be required. The other rates to be supported will be assumed to be lower rates and the base matrix will be augmented to generate these matrices; the base matrix is therefore the matrix utilized for the highest rate desired to be supported. This thesis' method for augmentation is as follows: the number of check node blocks is doubled and the permutation values inside the base matrix, in each column, are copied whilst appending NULL matrices as to fill the entire new matrix. To further elaborate, the first row of the base matrix will become the first two rows of the new, lower rate, matrix. The data or contents or permutation values will be copied only once and then paired with a NULL matrix block, denoted as '--', as to have one data become two. If one row is to become two then each column permutation value would need to become the new value for the new first or second row leaving the other to be the NULL matrix block.

A similar approach to construction, termed 'matrix expansion' is detailed in [49], yet in [49] the number of variable nodes and check nodes are both scaled by a constant factor ρ and NULL matrices are inserted in the newly created gaps. This is a good approach when rates of other forms need be supported; the focus

of this thesis is however matrices of form R-1 / R due to the case study implemented.

An illustration of this thesis' construction method for lower rate matrices' construction from a single high rate base matrix is shown below, for the general case, in Figure 1.12. Note how the code block length is constant, a significant variation to the expansion method provided by [49].

To clarify an important fact, it should be noted that only one of the matrices is utilized at a time, i.e. a decoder only operates at one rate at a time.

Below is the base matrix of rate $\frac{R-1}{R}$ *above the matrix of rate* $\frac{\left\lfloor\frac{R}{2}\right\rfloor-1}{\left\lfloor\frac{R}{2}\right\rfloor}$, *where R is a temporary variable to demonstrate a relative rate; note that the two earlier expressions involving R are of the same form.*

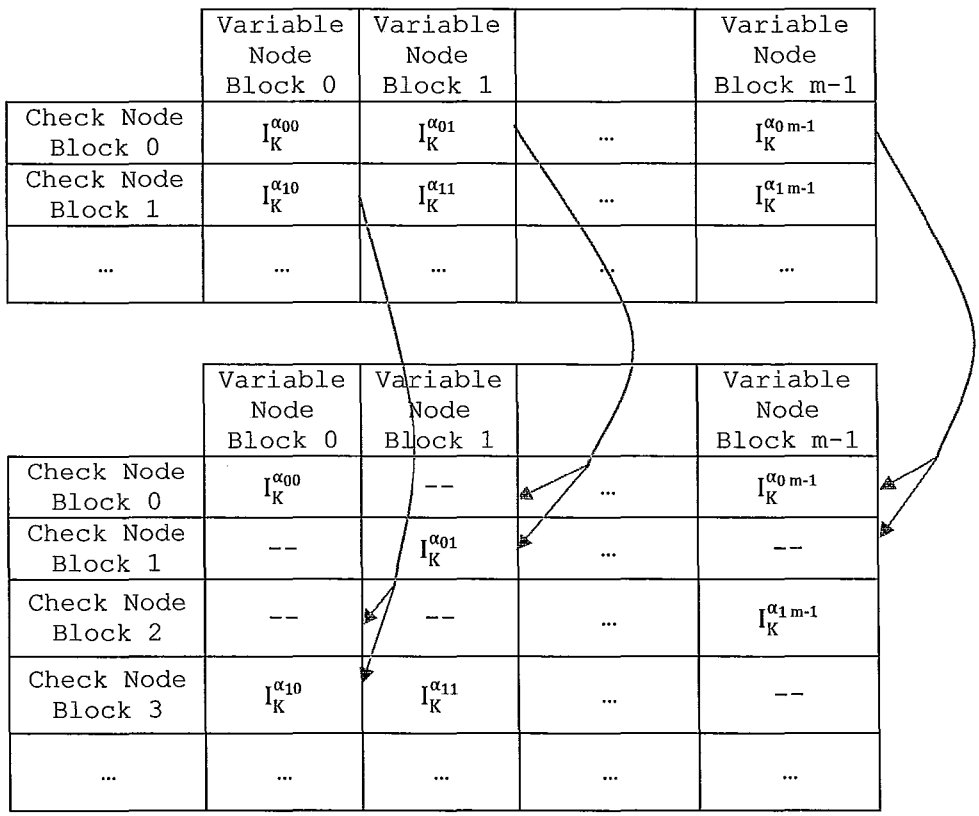


Figure 1.12: Illustration of matrix construction method. Note that arrows do not show all possible data traversal as to simplify the illustration.

It should be noted that the decision of the NULL matrix location (i.e. the first or second row in the new matrix if coming from the first row in the base matrix) need only be made as to have the degrees of the two new rows be equal.

The row degree is the count number of permutation values across the entire row due to said values' representations, or simply due to the fact that each permuted identity matrix has only one '1' in each row regardless of the permutation count; '-', or NULL, matrices do not contribute to the degree.

This method of construction apparently allows for a hardware efficient design as it simplifies interconnection changes across different rates. A similar design process is used by the case study of this thesis, 802.15-3c, and is further augmented to include what some could call puncturing; pseudo puncturing as termed by this thesis. In pseudo puncturing the goal is not to increase rate but merely to decrease interconnection density, as a result some values are not copied into the respective lower rate matrices and are simply replaced by NULL matrices. Many standards could be slightly augmented as to support these matrix properties such as DVB-S2, or WiMAX, as they use quasi-cyclic codes in the base matrix construction.

It should be noted that the form of the rate, i.e. R / R-1, is a result of the number of check nodes doubling across matrices. To support alternative rates could possibly be done by having some values propagate to two locations and other values propagate to a different integer number of locations. This detail was not explored as the case study had matrices of compatible rate form.

In the following section this thesis' analysis of prior art and related work will be detailed with sub sections: first, design space (section 2.1) wherein the basics of LDPC decoder design space is described along with a cursory comparison of some prior art; second, architecture art (section 2.2) wherein a detailed analysis of three state of the art architectures is provided.

Chapter 2

2 - Prior Art and Related Work

LDPC decoders have many design parameters that may lead to device improvements or shortcomings. In the first section of the following review of previous work, some key aspects of design parameters will be addressed and state of the art designs will be compared in a cursory manner. Second, a detailed analysis of three state of the art architectures is presented.

2.1 - The Design Space

Of all the prior art explored in this review only one work attempted a detailed comparison of different works across multiple design parameters [3]. When isolating the main comparison points, or attributes, a table was compiled to describe the design space as seen by [3]. This table covers many key design parameters such as: LDPC code design, the decoding algorithm, number of

iterations, data quantization, and architecture parallelism. The table below (Table 2.1) is a slightly augmented version of the table from [3].

Design Parameters	VLSI Parameters		Communications	Service Parameters	
	Area	Throughput	Performance	Block Length	Code Rate
LDPC Code	More edges increase RAM area. Higher code rate flexibilities increase logic area.	More edges decrease throughput. Lower edge/(Rate·VN) -ratio increases throughput.	Irregular LDPC codes perform better than regular LDPC codes.	Smaller block length reduces number of edges and irregularity.	Higher code rates reduce irregularity.
Algorithm	Larger area allows for more optimal decoding algorithms.		Optimal algorithms perform better than suboptimal ones.	Smaller block lengths are more suitable for suboptimal algorithms.	High code rates are more suitable for suboptimal algorithms.
Iterations		Throughput is inverse proportional to the number of iterations.	More iterations increase communications performance.	Larger block lengths require more iterations.	Higher code rates require less iterations.
Quantization	Larger area allows for wider quantization.		Increased performance with higher quantization.		Higher code rates allow for smaller quantization.
Architecture Parallelism	Increased logic area by higher parallelism.	Throughput proportional to parallelism.	Parallelism can limit communications performance.	Larger block sizes allow for higher parallelism.	

Table 2.1: Important Parameters of the LDPC Decoder Design Space [3].

The prior art concerning this work will be analysed and categorized loosely with respect to the table above in a row by row approach where the design parameters will be addressed in the order: LDPC code structure, algorithm design, iteration count, quantization size of the internal data, and finally architecture parallelism.

Regarding the portion of the design space that is the LDPC code itself one may note that, as more edges are added or as the parity check matrix density increases, area increases and throughput decreases [3]. Most of the previous work in the scope of this thesis does not focus on altering the LDPC code. The exception to this is [28], wherein [28] demonstrates a means of code construction as to alleviate interconnection complexity. Aside of [28], the prior art in the scope of this thesis that do alter the LDPC code, merely perform row operations on the code and do not alter the density (such as in [6], [25], and [40]). The purpose for altering the code through row operations in this way is to alleviate the layered decoding scheme; this is done by [6], [40]. Recall that the nuances between the different decoding schemes are described in section 1.2.3. Not all architectures are produced for a specific LDPC code, in fact in [29] a completely programmable architecture is provided.

Concerning the portion of the design space which focuses on the algorithm, a good parameter for comparison across the prior art is realized. There are three main types of Belief Propagation based decoding algorithms that are utilized by the prior art: the standard Sum-Product algorithm (SPA), the Min-Sum algorithm (or equivalently the Uniformly Most Powerful algorithm, UMP), and the λ-Min algorithm. Again, all the formulae for the above calculations are in the LDPC decoding section of this thesis. To continue, already depicted by Table 2.1

the SPA algorithm does result in a higher area due to algorithmic complexity, yet it also has high performance. The optimal performance relative to area seems insufficient as to justify the use of SPA for any of the architectures amongst the prior art; none of the prior art utilizes the more complex formulae without a simplification that results in performance degradation except for [47]. An older architecture, [47] is an architecture in which the only SPA + layered decoding scheme is utilized; area is sacrificed for throughput.

Some of the prior art does in fact attempt an altered, or simplified/reduced, flavour of the SPA that reduces area as to improve performance relative to area; a collection made up of [31], [34], [36], and [37]. Many of the prior arts (i.e. [4], [6], [7], [14], [18], [40], [43], and [48]) use the Min-Sum (or UMP) decoding algorithm or a scaled flavour of the same algorithm. Such a scheme yields a good area to performance relation and is what is used by this thesis. The Min-Sum decoding algorithm can also be further simplified into the λ-Min algorithm by using the value-reuse property of the Min-Sum algorithm; utilized by [12], [13], [17], [24], [35], and [41]. Algorithm properties and descriptions can be found in section 1.2.3.

Regarding iteration, or better said iteration count, meaningful comparison across the prior art is moot, what with [34] implementing both a two phase approach and layered approach, along with [14], [36], [37], [40], and [48]

implementing a two phase approach and [4], [6], [7], [12], [13], [15], [17], [18], [24], [43], and [47] implementing a layered (single phase) approach, the iteration count can differ by a factor of two, *ceteris paribus*. This factor is due to the inherent algorithmic structure of the layered approach and the details of the layered approach scheme can be found in the LDPC decoding section of this thesis (section 1.2.3).

The parameter that can drastically affect area and performance is quantization [3]. Not many of the papers show results for multiple quantizations as this could result in the need of complete redesign of the architecture. The common quantization used amongst the prior art seems to be six bits (1 sign, 3 or 4 integer, and 2 or 1 fraction), used by [4], [7], [13], [24], [34], [35], [36], [37], [40], [43], and [48]. Whereas: 8 bits (1 sign, 4 integer, and 3 fraction) are used by [41]; 4 bits (1 sign, 2 integer, and one fraction) by [6], and [47]; and 5 bits (1 sign, 2 integer, and 2 fraction) by [12], [14], [17], and [18].

Architecture parallelism is an influential parameter, it can affect whether or not a scheme is feasible. What is entailed in adjusting architecture's parallelism is the scaling of how many processing engines are used. In other words, architecture's parallelism is simply how many parallel processing nodes are used to calculate the formula of interest.

For example if there are 672 nodes to be processed, then updates may be processed via 672 consecutive uses of the same single engine; the least parallel case. The 672 nodes may also be processed by 672 parallel engines in one use; the most parallel case. All the different works commented on thus far have different levels of parallelism, and in most schemes the parallelism is addressed in a manner as to be just parallel enough as to produce the desired throughput. Otherwise the parallelism is constrained by the area, as parallelism has a significant and direct effect on area and area is expensive. Other art such as [29] even have the parallelism as a programmable design constraint.

In the following section 2.2 three basic architectures will be compared. The variation amongst the architectures may seem minimal yet the differences are significant.

2.2 – Prior Work on LDPC Decoders

In this section two architectures from [36] and a single architecture from [14] are presented as prior art and discussed. Below in Figure 2.1 is the first architecture to be discussed from [36].

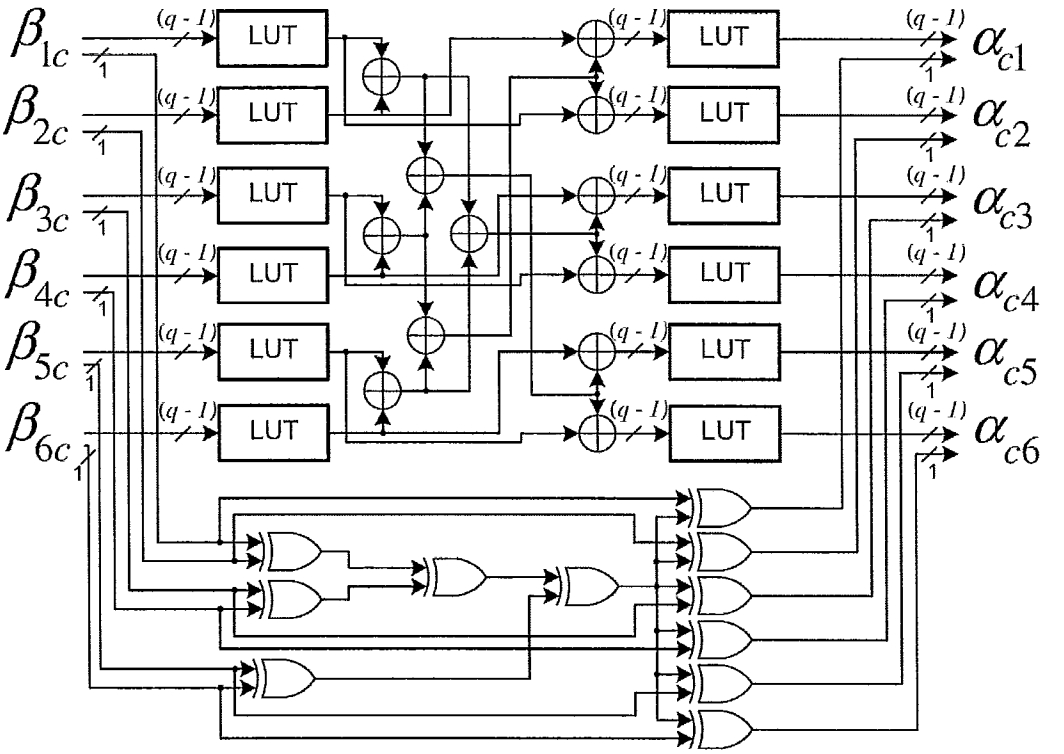


Figure 2.1: A check node architecture from [36] which utilizes SPA as the decoding algorithm.

In the above figure, the SPA algorithm (Equation 1.1) is implemented for a check node of degree six. The inputs are for the messages from the six variable nodes connected to the particular check node c, and message quantization is q bits. The messages from the six variable nodes (the β values) connected to check node c are passed in, and the messages from c back to the six connected variable nodes are outputted (the α values). The circled plus blocks are simply addition blocks. The Ψ function (log(tanh(x/2)) in Equation 1.1 is implemented using Look-up-tables, or LUTs. Each of these LUTs has a table of possible outputs uniquely and consistently determined by the input look up address. For example, regarding a LUT with four address bits, sixteen, or 2^4, different binary outputs can be programmed as address dependant values. Implementing LUTs in parallel (i.e. the LUTs all share the same address inputs and the LUTs outputs are concatenated) allows for multi-bit output; this is what is implied when LUTs have multi-bit outputs. Hence the degree of inputs for a LUT need not have any relation to the degree of output. In the case in the earlier figure, Figure 2.1, the LUT inputs are q-1 bits wide with outputs of the same width. These are later treated as new values that will propagate through the circuit. The formula implemented by these LUTs in Figure 2.1 is: $\Psi(x) = \log(\tanh(|x/2|))$ [36], it is clear that Ψ may be substituted in the update formula for check nodes in Figure 1.5.

The gains of this approach are reflected in the accuracy of the calculations; see section 1.2.3 for more details on why SPA has such accuracy. Regarding the accuracy of LUTs is an involved discussion that covers much theory and thus the analysis of this accuracy is out of the scope of this thesis; please refer to [36] for more case specific details.

The drawbacks of the approach illustrated in Figure 2.1 are reflected in the needed area relative to performance. In fact, [36] does recognise this, and as a result introduces the augmented version of the architecture in which reduced-look-up-tables, RLUTs, in tandem with compression units, COMPs, are implemented as to maintain performance and simultaneously reduce area. The second architecture to be discussed, the architecture for this augmented flavour is shown below in Figure 2.2.

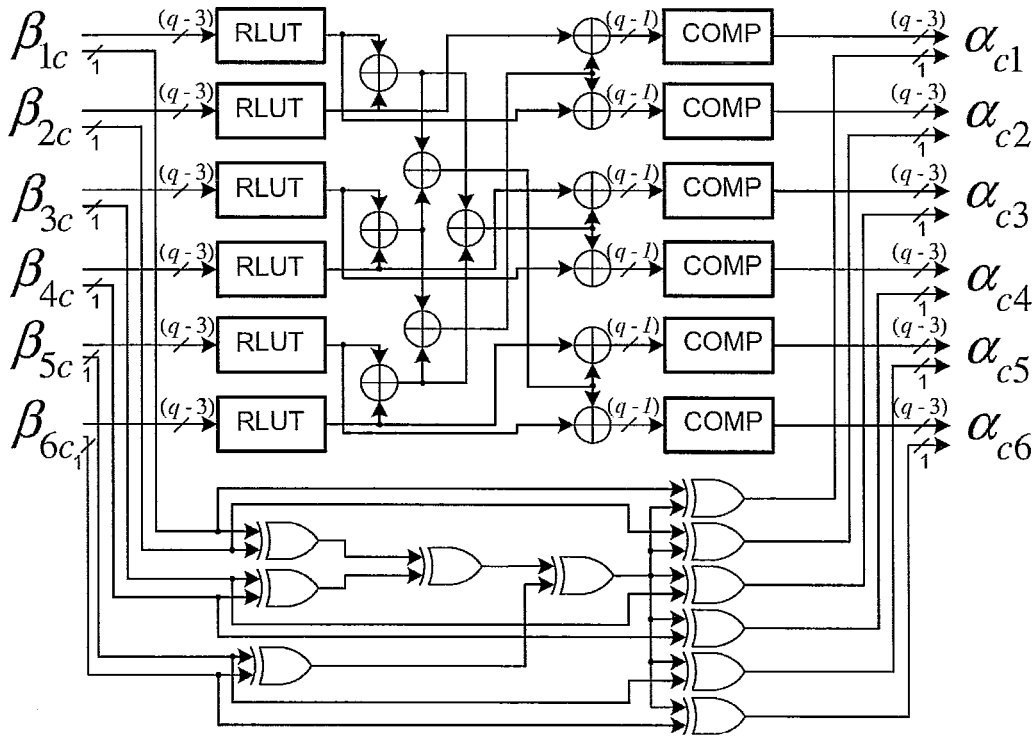


Figure 2.2: A check node architecture from [36] which utilizes an augmented SPA with Reduced-LUTs and Compression Units.

The RLUTs result in acknowledgeable savings in area (described in [36]), yet the mathematical reasoning of the RLUTs may be unclear; a figure from [36] is provided for clarification below in Figure 2.3. Note the columns showing the variation in quantization. What should be noticed is the number of different possible outputs for each quantization scheme. Regarding the 'Uniform quan' (uniform quantization) and corresponding 'Decimal value' (simply the function

Ψ(LUT input) result in decimal) columns, one can clearly see the standard result from using a LUT to implement Ψ.

LUT input	LUT output			
	Decimal value	Uniform quan.	Variable quan. I	Variable quan. II
00000	3.500	11100	11100	11100
00001	2.375	10011	10011	
00010	1.875	01111	01111	01111
00011	1.500	01100	01100	
00100	1.250	01010	01010	01010
00101	1.125	01001	01001	
00110	1.000	01000	01000	01000
00111	0.875	00111	00111	
01000	0.750	00110	00110	
01001	0.625	00101		00101
01010	0.500	00100	00100	
01011	0.500	00100		
01100	0.375	00011	00011	
01101	0.375	00011		00011
01110	0.375	00011	00010	
01111	0.250	00010		
10000	0.250	00010		
10001	0.250	00010	00010	
10010	0.250	00010		
10011	0.125	00001		00010
10100	0.125	00001		
10101	0.125	00001	00001	
10110	0.125	00001		
10111	0.125	00001		
11000	0.125	00001		
11001	0.125	00001	00001	
11010	0.125	00001		
11011	0.125	00001		00000
11100	0.000	00000		
11101	0.000	00000	00000	
11110	0.000	00000		
11111	0.000	00000		

Figure 2.3: A figure from [36] to clarify the reasoning behind the RLUTs.

What is also apparent is the repetition of outputs. The output '00001' (or 0.125 in decimal) is outputted for nine different input addresses; redundant. The column 'Variable quan I' shows how some redundant outputs can be dropped,

with some outputs lost as well (such as 00101, 0.625). What should be clarified is the number of outputs in each column is always a power of two. This is due to the inherent nature of the input, being a binary vector of however many bits, $2^{\text{number of input bits}}$ outputs should be generated; repetition may still occur amongst the outputs. As a result some values are lost and others still repeat; overall redundancy, however, is still reduced. To support the 16 outputs in the 'Variable quan I' column would roughly require half the area as that required to support the 'Uniform quan' column. This assumes only LUTs are used, as logic can be built with other base blocks such as NANDs. The same relation of needed area may be made between the 'Variable quan I' and 'Variable quan II' columns.

This leads to a complication, the five-bit input shown in the 'LUT input' column seems to also be the three-bit input for the eight outputs of the 'Variable quan II' scheme (or the four-bit input for the sixteen outputs of the 'Variable quan I' scheme). This is a misconception, the table is simply indicative of the Ψ function remapping. The eight values in the 'Variable quan II' column need a three-bit driver and this is clear when regarding Figure 2.2 (q is six in this example, i.e. one sign bit and 5 fractional + integer bits). Note how the bitwidths are smaller at the input and output of the check node calculation block. This is because the values outside the check node calculation are no longer the extrinsic

information of the nodes. As a result of the COMP blocks, the values are now compressed extrinsic information.

The COMPs can be thought of as merely a means of converting the q-1 bit wide results of the RLUTs down to the q-2 or q-3 (depends on variable quantization scheme I → q-2, and II → q-3) compressed inputs later expected by connected nodes; i.e. this same pairing of RLUTs at input and COMPs at outputs are in both the variable and check node architectures. To facilitate this pairing, the SPA formula was reorganized by [36] as to have the Ψ function calculation present in both check node and variable node update calculations. This allows all memory storage to be reduced by a factor of roughly two for the 'I' and four for the 'II' quantization schemes. The reorganized SPA formulae and original SPA formulae are shown below in Figure 2.4.

$$\alpha_{cv} = (-S_{cv})\,\Psi\left(\sum_{n\in N(c),n\neq v}\Psi(\beta_{nc})\right)$$

$$S_{cv} = \prod_{n\in N(c),n\neq v}\operatorname{sgn}(\beta_{nc})$$

$$\beta_{vc} = \sum_{m\in M(v),m\neq c}\alpha_{mv} - \frac{2Y_v}{\sigma^2}$$

$$\alpha_{cv} = -S_{cv}\cdot\sum_{n\in N(c),n\neq v}\Psi(\beta_{nc})$$

$$S_{cv} = \prod_{n\in N(c),n\neq v}\operatorname{sgn}(\beta_{nc})$$

$$\beta_{vc} = \sum_{m\in M(v),m\neq c}(-sgn(\alpha_{mv})\cdot\Psi(\alpha_{mv})) - \frac{2Y_v}{\sigma^2}$$

Figure 2.4: The SPA formulae reorganization utilized by [36].

In short, [36] seeks to reduce memory storage space and communication fabric density by compressing the messages stored and passed in the network. Such

an approach is quite novel yet not applicable to this thesis' work due to the thesis' use of the UMP rather than the SPA.

The architecture from [14] is now addressed as to show how design choices impact area. Regarding the internal circuitry in the check node update calculation block for a UMP based implementation it is obvious that these calculations are more simply implemented with a signed magnitude representation of internal values. An architecture for UMP based check node update calculation block is shown below in Figure 2.5; again note how the internal operation is the minimum function and is more easily implemented with a signed magnitude numerical representation (the architecture in the figure in fact expects a signed magnitude representation).

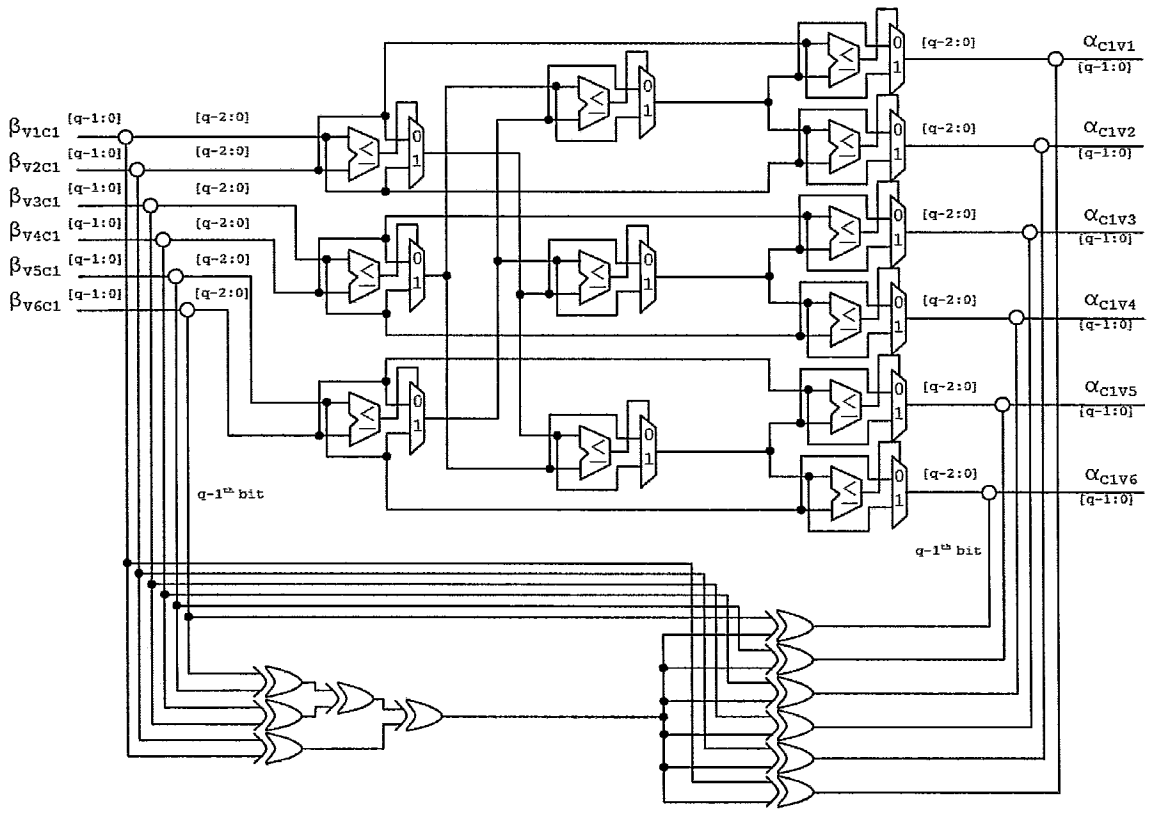


Figure 2.5: An example of a UMP based check node update calculation.

Alternatively, regarding a UMP based variable node update calculation block, the preferable numerical representation is 2's complement; refer to Figure 2.6 below, where internal calculations are additions which are suited for 2's complement numerical representation (the architecture in fact expects a 2's complement numerical representation).

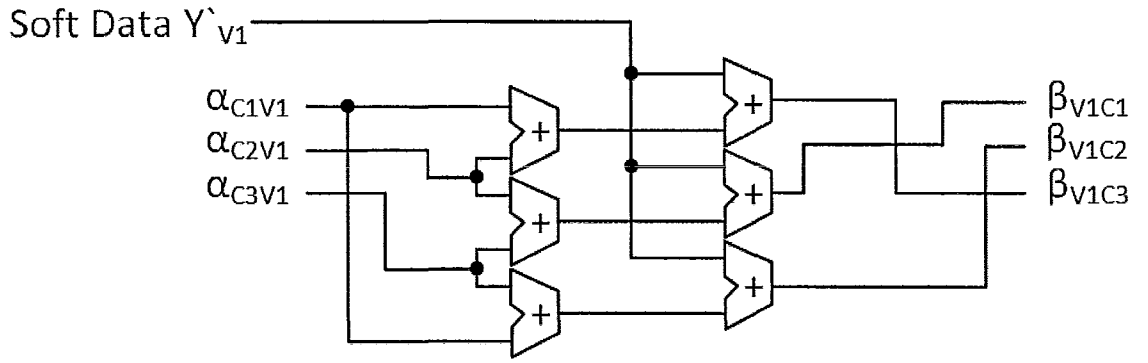


Figure 2.6: A UMP based variable node update calculation block, all wires are q bits wide.

When selecting the numerical representation implemented outside the node update blocks, it is important to understand that a conversion between representations will be needed in one of the calculation blocks; either inside the check nodes or variable nodes. To choose the check node as the block in which the conversion occurs is the design choice that [14] has made. The reason why this choice results in more area as opposed to the alternative is clear when concerning degrees of nodes. The figure for the architecture in [14] is below in Figure 2.7.

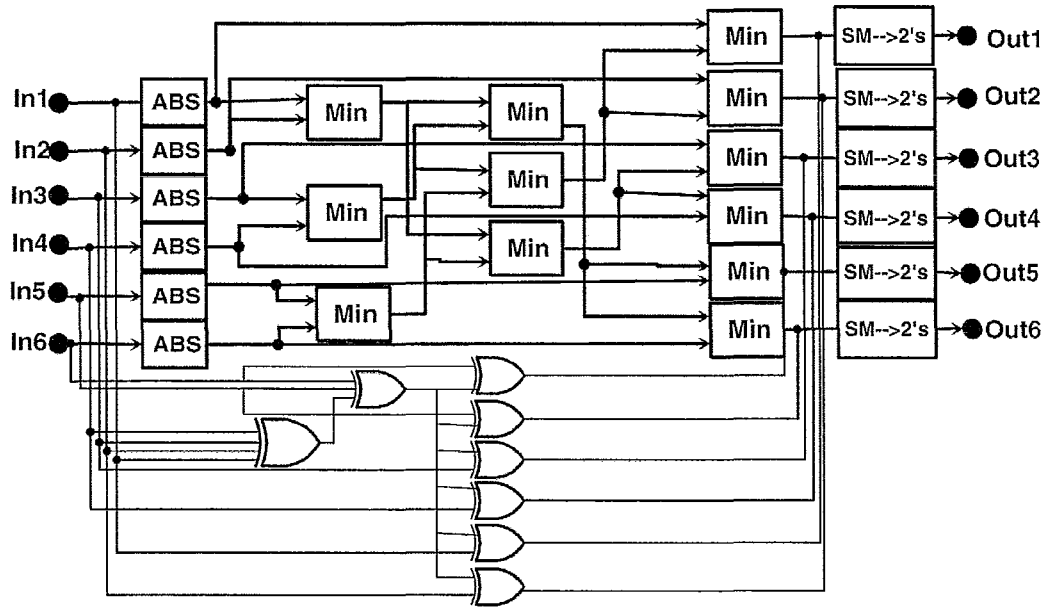


Figure 2.7: A Check node architecture from [14] which maintains numeric representation, outside the check nodes, in 2's complement.

If a particular node (variable or check) is of degree X then 2X conversion units would be required. X conversion units would be required for the X inputs to be converted into the preferable numerical representation, the internal calculation would then be performed, and then X conversion units would be required to convert the X outputs. If the average degree of check nodes was equivalent to that of variable nodes, the decision of implementing the conversion units in the check nodes or variable nodes would be trivial; this is not the case in practical implementation of codes constructed in section 1.2.4.

In fact the average degree of check nodes in the higher rate codes, such as the case study of this thesis (namely the parity check matrices for 802.15-3c), can be as much as an order of magnitude in decimal (30.5 for check and 3 for variable).

In general, recall that the rate of an LDPC equates to one minus the ratio of check nodes to variable nodes. Thus with any feasible code, i.e. rate less than one and positive, there are more variable nodes than check nodes. With the number of connections between nodes constant, i.e. the number of messages passed back and forth is constant, along with the presence of more variable nodes than check nodes, it is obvious that the average degree of variable nodes is less than that of check nodes. Hence, having 2's complement as the numerical representation of the extrinsic values passed between the variable and check nodes across iterations of the UMP algorithm results in an architecture with more conversion units than necessary.

In the following section this thesis' contribution will be portrayed with sub sections: first, preliminary concepts (section 3.1) wherein the basics of LDPC codes will be covered with the focus on the necessary information as to comprehend section 3.2; second, node nesting (section 3.2) wherein the inherent nature amongst matrices constructed according to section 1.2.4 is described; third, architectures for node nesting (section 3.3) wherein this thesis will present

comparison of both intelligent and insufficient architectures that provide support for the node nesting; and fourth, implementation (section 3.4) wherein the design process adopted by this thesis along with the corresponding results are provided.

Chapter 3

3 – A New VLSI Architecture and Its FPGA Implementation

One important concept to realize about this work is the generality in its applicability. To elaborate, assuming that the set of parity check matrices to be supported are constructed in the same manner as stated earlier in section 1.2.4, i.e. using a seed matrix and expanding, this work can be applied to provide a single architecture that can support the created set of matrices in addition to the base matrix; a specific example will be used to clarify this point later in this section.

3.1 - Preliminary Concepts

When regarding the formula, or algorithm, for this work's chosen LDPC decoding heuristic, one can be overwhelmed by the seemingly complex structure. Recall the chosen heuristic is the uniformly-most-powerful belief propagation (UMP-BP) algorithm which uses extrinsic information; this point will be clarified

later in this section. We show this below for the check-node-update-calculation (α_{cv}), variable-node-update-calculation (β_{vc}), and the hard-decision-vector-component-calculation (β_v).

$$\alpha_{cv} = \left\{ \bigoplus_{n \in N(c), n \neq v} \text{SignBit}(\beta_{nc}) \, , \, \min|\beta_{nc}| \right\} \quad \text{(a)}$$

$$\beta_{vc} = \frac{2Y_v}{\sigma^2} + \sum_{m \in M(v), m \neq c} \alpha_{mv} \quad \text{(b)} \qquad \beta_v = \frac{2Y_v}{\sigma^2} + \sum_{m \in M(v)} \alpha_{mv} \quad \text{(c)}$$

Equation 3.1: The formulae for the check-node-update-calculation (a), variable-node-update-calculation (b), and hard-decision-vector-component-calculation (c).

In the above formulae the subscripts c and v along with the functions N(check node c) and M(variable node v) are of key importance when trying to understand which node instance is meant to be processed/calculated. The c subscript refers to the check node ID/instance, and the v subscript to the variable node ID/instance. The function N() has an input argument of a particular check node (c) and returns the set of variable nodes connected to the check node input argument. Similarly, the M() function has an input argument of a particular

variable node (v) and returns the set of check nodes connected to the variable node argument. Thusly, the set $\{n \in N(c), n \neq v\}$ simply equates to the set of variable nodes that are connected to check node c with the exclusion of variable node v, and the set $\{m \in M(v), m \neq c\}$ simply equates to the set of check nodes that are connected to variable node v with the exclusion of check node c. Y_V is the soft data from detection for variable node v and σ is the noise standard deviation of the AWGN channel in which the message is transmitted.

Another key point to finalize before compiling the earlier statements into a comprehensive explanation is this work's unique notation in the node formula (Equation 3.1-a). The {} brackets are used to represent a concatenation of two binary words; i.e. {a,b} produces the new word ab, or {0,101} produces 0101. The ⊕ symbol refers to the XOR or modulo-2 addition operation over all the 1-bit arguments in the following brackets. The SignBit() formula returns a one if the argument is negative and a zero otherwise. It should be noted that said argument's value is assumed to be represented in a Signed Magnitude format, therefore the SignBit() formula is a free operation in hardware as it is just the selection of the MSB. The min function simply returns the minimum value in the set described by the bounds on the min function which is $\{n \in N(c), n \neq v\}$ in the above case.

With all that, we may continue on to the example-driven explanation of the formulae as to later produce the simplified understanding of the formulae; important to grasp well as to be able to see the hidden simplicity utilized by this work.

Concerning a particular check node, a particular output is derived from all the inputs except said output's input counterpart. For example, regarding a particular check node, C1 in this example, connected to variable nodes: V1, V2, V3, V4, V5, and V6, six output values will need to be produced from six inputs. Namely outputs: α_{C1V1}, α_{C1V2}, α_{C1V3}, α_{C1V4}, α_{C1V5}, and α_{C1V6} from inputs: β_{V1C1}, β_{V2C1}, β_{V3C1}, β_{V4C1}, β_{V5C1}, and β_{V6C1}. The key to perceiving the subtle simplicity used by this work is found when focusing on a single output, for the sake of argument and this example, α_{C1V1}. The MSB of α_{C1V1} equals the XOR of: SignBit(β_{V2C1}), SignBit(β_{V3C1}), SignBit(β_{V4C1}), SignBit(β_{V5C1}), and SignBit(β_{V6C1}), and the magnitude of α_{C1V1} equals the min of: $|\beta_{V2C1}|$, $|\beta_{V3C1}|$, $|\beta_{V4C1}|$, $|\beta_{V5C1}|$, and $|\beta_{V6C1}|$. Note how the output α_{C1V1} is in no way directly related to β_{V1C1}, this is what is meant by 'output is derived from all the inputs except said output's input counterpart' and is also implied by the fact that UMP-BP propagates the nodes' extrinsic information to each update value. Extrinsic information is the information found outside the node of concern, i.e. in the node's connected neighbours. The circuit to calculate the above example of a check-node-update-calculation is

shown below in Figure 3.1. Note q is the word length, solid circles are connections, and open circles are wire joiners/splitters (see labelled word lengths).

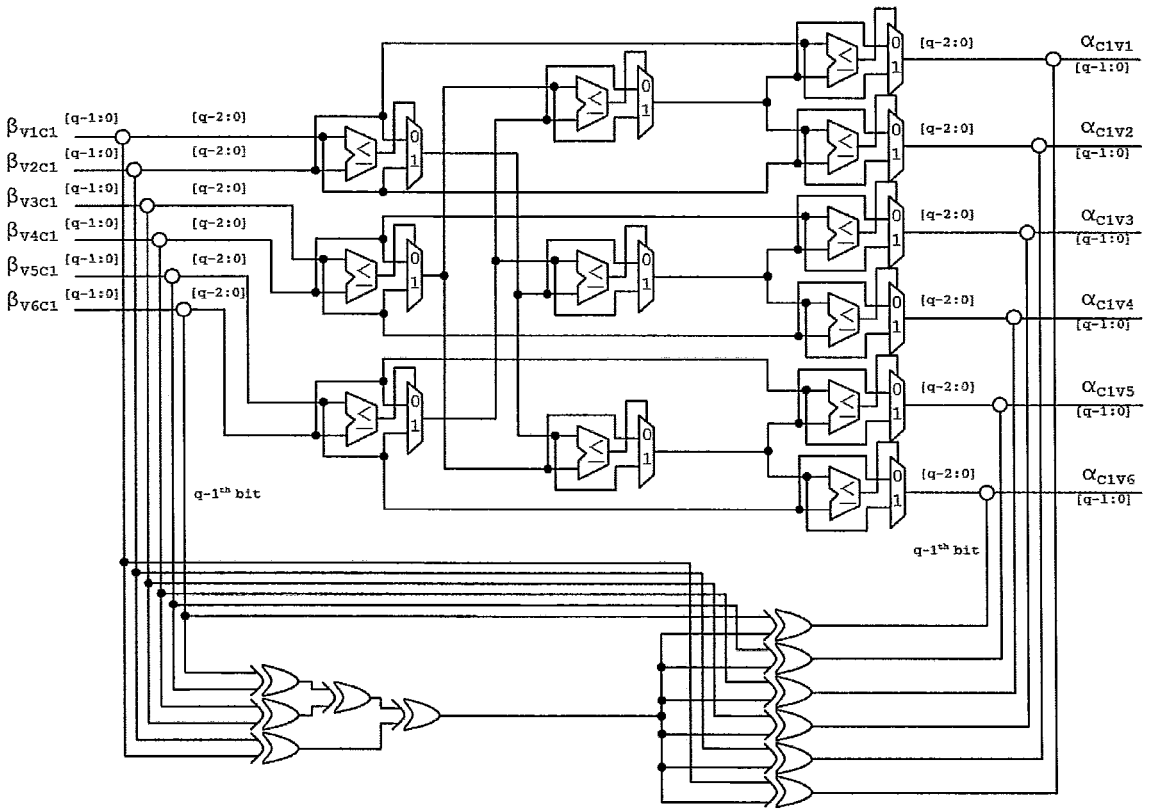


Figure 3.1: An example check-node-update-calculation processing block architecture.

Concerning oneself with a particular variable-node-update-calculation (Equation 3.1-b) and the accompanying hard-decision-vector-component-calculation (Equation 3.1-c), the same sense of simplicity may be derived as earlier for the check-node-update-calculation. For variable node V1 connected to

check nodes: C1, C2, and C3; three output values will need to be produced from four inputs, namely outputs: β_{V1C1}, β_{V1C2}, and β_{V1C3} from inputs: α_{C1V1}, α_{C2V1}, α_{C3V1}, and $Y`_{V1}$ (i.e. $2 * Y_{V1} / \sigma^2$, the scaled soft data input with value assumed to be represented in 2's complement). Focusing on a single output of this system, for the sake of argument and this example, β_{V1C1}, the value of β_{V1C1} equals the sum of: α_{C2V1}, α_{C3V1}, and $Y`_{V1}$. Again note how the output β_{V1C1} is in no way directly related to α_{C1V1} and that for the hard-decision-vector-component-calculation all that needs to be done is the incorporation of α_{C1V1} (choosing the sign bit incurs no cost in hardware as it is merely a selection of the most significant bit (MSB)).

Below in Figure 3.2 for the above example focused on variable node V1, the correlation between the hard-decision-vector-component-calculation (denoted as HD where each of the one-bit HDs together form the HD vector) and the variable-node-update-calculation can be exploited by merging the two calculations into a single architecture.

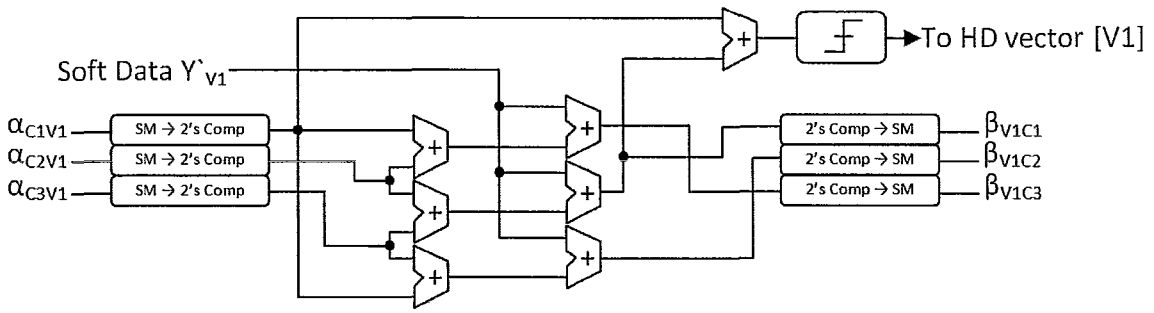


Figure 3.2: The hard-decision-vector-component-calculation and variable-node-update-calculation in a single architecture as to reuse hardware.

3.2 - Node Nesting

With all the above now established, this work's contribution may be elaborated. The reason why there is the earlier requirement of matrix construction is to allow for the occurrence of nesting nodes; this can be exploited as to reuse hardware and thusly save area. Considering the figure of the matrices from the case study of this work (Figure 3.3), i.e. the parity matrices for 802.15.3c [16], along with the corresponding arrows, the nesting of nodes is apparent when focusing on where to the blocks from the rate 7/8 matrix translate, or copy, in the lower rate matrices.

For example, noting the first-column first-row, in the rate 7/8 matrix, the value 0 (meaning zero permutations of the 21x21 identity matrix as earlier stated in section 1.2.4) translates to either the first or second row of the 3/4 rate matrix (here the 1^{st}), leaving the other block slot (here the 2^{nd}) filled with zeros. In other words, if a N-row base matrix is expanded to a 2N-row matrix, the N^{th} row of the base matrix becomes the $2N^{th}$ and $2N^{th}$-1 rows of the new lower rate matrix.

It should be noted that the row degrees are paired, i.e. the two rows in one matrix (the first two rows in the rate 3/4 matrix) that are nested in a single row of the higher matrix (the 7/8 matrix) both have the same degree (not equal to the degree of the single row, and in fact should be at most half of the degree of the

single row due to the pairing); this is depicted in a table to the right of Figure 3.3. Note that the matrices are obfuscated as to protect the intellectual property and this does not result in any loss of information directly required to understand this thesis' work.

To maintain these paired degrees not all the values translate, and as a result the constructed node macro-blocks are not only comprised of just two sub-blocks but three. Figure 3.4 uses this concept of nested nodes to construct the macro-blocks that act as reconfigurable nodes for the multiple rates, and shows the excess nodes that result from the values that do not translate. The specifics of the pseudo puncturing which is used to drop values and the rational behind utilization of this pseudo puncturing is outside of the scope of this thesis and is not addressed (see [39] for a detailed explanation on LDPC codes and their construction).

To clarify an important fact, it should be noted that only one of the matrices is utilized at a time, i.e. a decoder only operates at one rate at a time. Thus to have nested nodes is beneficial as module reuse normally results in area savings.

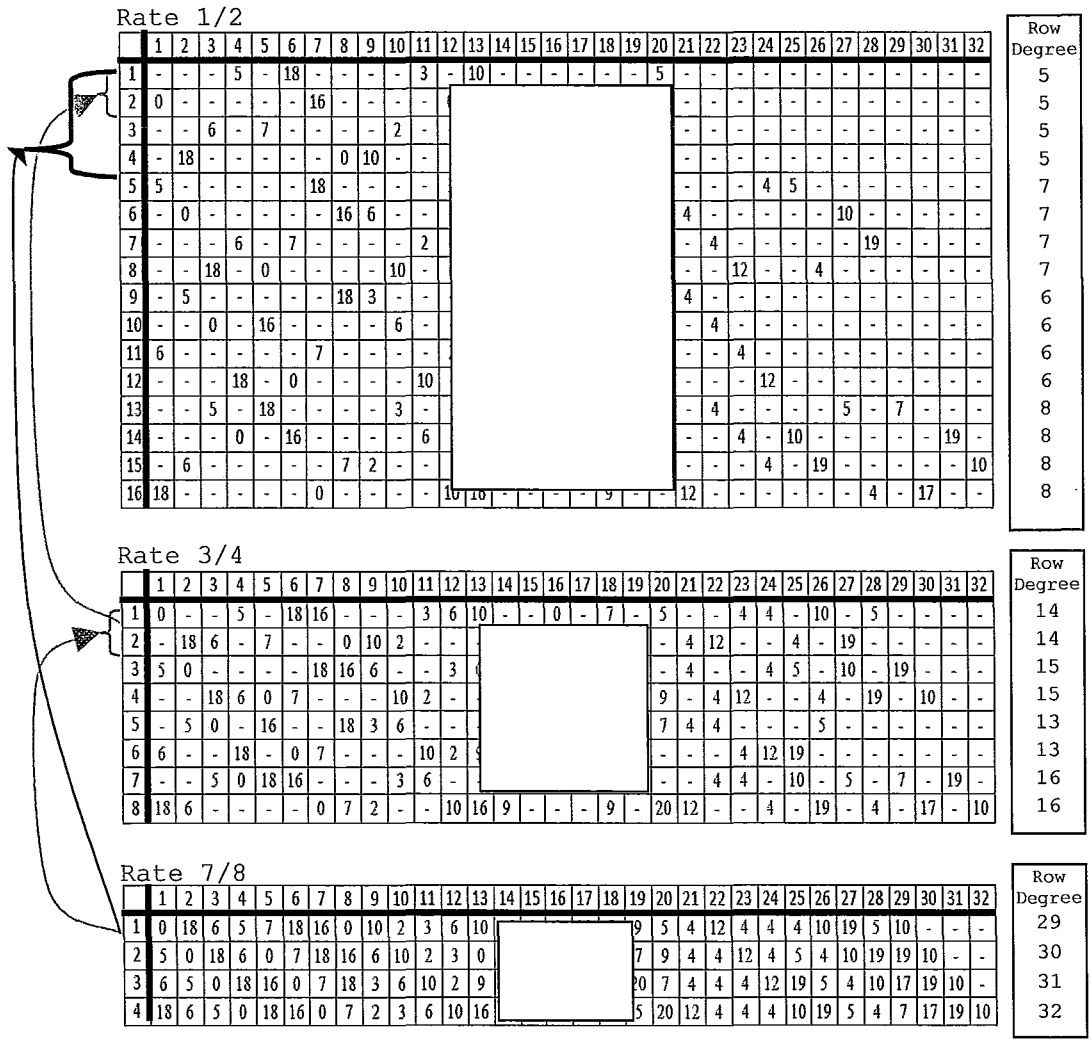


Figure 3.3: The matrices used in this work's case study [16].

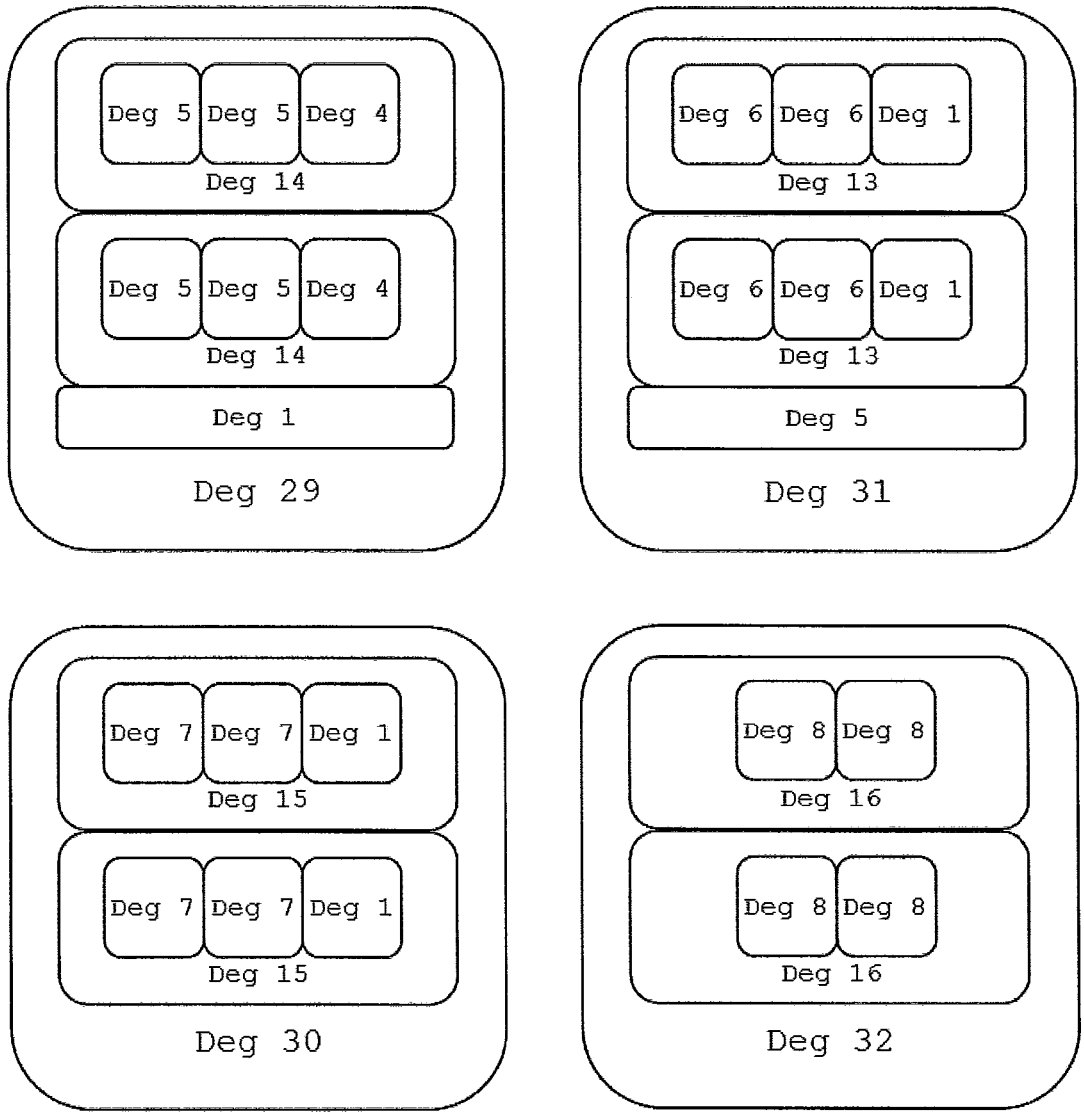


Figure 3.4: The depiction of the macro blocks designed for the case study of this work.

To portray by example, the degree-29 macro-block is the single macro-block needed to support the first row of the rate 7/8 matrix; i.e. one degree-29 block to support the 29 degrees needed. Note that by degree the number of input

data is quantified; Figure 3.1 is of degree six. This macro-block is also the only macro-block needed to support the first two rows of the rate 3/4 matrix, and is also the only macro-block needed to support the first four rows of the rate 1/2 matrix. Regarding the sub-blocks which may seem unnecessary, the reader must understand that they are merely a means of providing the degree support needed, and in fact come from the matrix construction; they are a result of the values that were not traversed between matrices as discussed earlier. For example, removing the degree-1 sub-block inside the first level of the degree-29 macro-block would make the degree-29 macro-block an effective degree 14+14=28 and thusly would no longer support the 29 degrees required. The other reason why the degree-29 macro-block has the degree-1 sub-block is because there is only one value not traversed from the rate 7/8 matrix's first row to the rate 3/4 matrix's first two rows. Similarly, removing the degree-4 sub-block from one of the degree-14 sub-blocks would make that degree-14 sub-block only a degree 5+5=10. The other macro-blocks have the same general behaviour/organization as that of the degree-29 macro-block and, because of this similarity, will not be elaborated.

It should also be noted that the variable nodes (i.e. the columns in the matrix) are connected to the same check nodes in all the matrices with some simple exceptions handled with intelligent masking discussed later. This allows

the same variable nodes to be connected to the same macro-blocks as rate changes, an important attribute of these matrices' construction. Multiplexers for redirecting node connections at the check-node-update-calculation macro-blocks' inputs and variable-node-update-calculation blocks' inputs are not needed, and the simple task of intelligent masking of inputs that do not translate is all that is needed to be added; Figure 3.5 shows the general idea of this intelligent masking below.

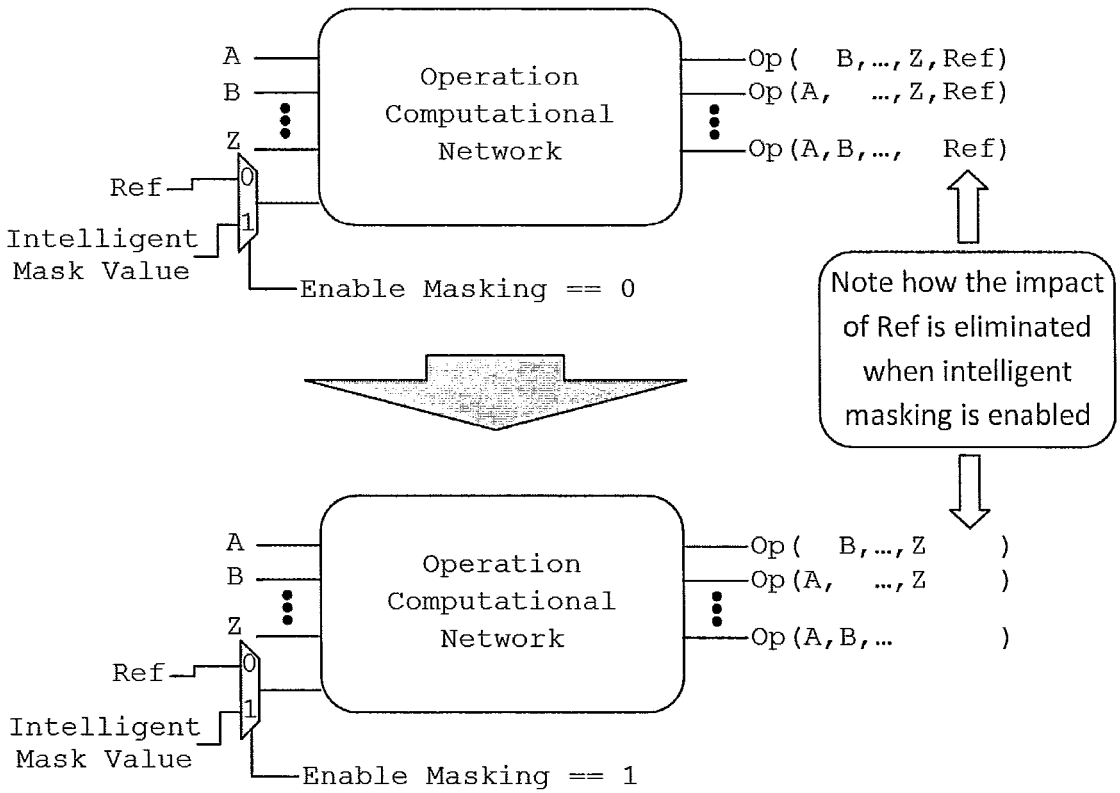


Figure 3.5: Intelligent Masking Depiction.

Intelligent masking is elaborated in the following: regarding check-node-update-calculation, an input of the largest possible positive number will not affect the outputs. This is because each output is a minimum function of the respective inputs, so the input of the maximum possible value will have no effect; the sign bit has the same situation with an input of 0. To further express in other words, the outputs' MSBs are derived from an XOR network of sign bits and a value of 0 XOR x will equate to x thus the sign bit is not affected. In the case of the magnitude of the output, the value is derived from the inputs' minimum, thus an input of the maximum possible number would not affect the output. Similarly, regarding the variable-node-update-calculation, an input of zero will not affect the outputs as x+0=x. Using this intelligent masking allows for simpler connections to the check-node-update-calculation-macro-blocks, and for variable-node-update-calculation blocks' hardware to be shared across matrices. This attribute of lining up also leads to the reason why this creation of macro-blocks is only done for the check-node-update-calculation.

A similar pattern exists when regarding the variable-node-update-calculation, although to exploit this pattern would not result in a significant gain in organizational simplicity. The reason why is clear when considering the case study and the fact that the number of different degrees that need to be supported for the variable-node-update-calculation is only four, and in fact the number of

degree-4 variable nodes is much higher than the number of degree-1 or 2 or 3 nodes. Recall the number of variable nodes is larger than that of the number of check nodes (see end of section 2.2). Thus with the number of messages passed between nodes constant, the degree of the check nodes is higher on average than that of the average degree of variable nodes (see end of section 2.2). Also to have a nested situation as do the check nodes, the variable nodes would have to increase in count and they do not. Thus a macro-block system is not implemented for the variable-node-update-calculation.

3.3 - Architectures for Node Nesting

Next to be clarified is the means of creating these check-node-update-calculation macro-blocks. The basic architecture shown in Figure 3.1 would not support merging, i.e. the joining of sub-blocks. To be able to create an architecture that could support merging, we need to first readdress the outputs of the check-node-update-calculation macro-blocks. Recall the earlier example with check node C1, connected to variable nodes: V1, V2, V3, V4, V5, and V6, where the MSB of α_{C1V1} equals the XOR of: SignBit(β_{V2C1}), SignBit(β_{V3C1}), SignBit(β_{V4C1}), SignBit(β_{V5C1}), and SignBit(β_{V6C1}), and the magnitude of α_{C1V1} equals the min of: $|\beta_{V2C1}|$, $|\beta_{V3C1}|$, $|\beta_{V4C1}|$, $|\beta_{V5C1}|$, and $|\beta_{V6C1}|$. Consider that we intend to add a single degree to this degree-6 node making an effective degree-7 node. The output $\alpha_{C1V1}^{\text{new-degree7}}$ would equal, if the reader would recall the earlier notation used to express the formula for the check-node-update-calculation, {SignBit($\alpha_{C1V1}^{\text{old-degree6}}$) XOR SignBit($\beta_{Deg1}$) , min($|\alpha_{C1V1}^{\text{old-degree6}}|$,$|\beta_{Deg1}|$)}.

Thus, one may think of an output as {XOR (of all the nodes of interest MSBs), min (of all the nodes of interest magnitudes)}. Hence, to provide a reference that would be {XOR (of all the nodes' MSBs in another block), min (of all the nodes' magnitudes in another block)} allows the incorporation of the other

block's information. This reference may then be treated as another degree to incorporate.

Using that fact, we may allow two nodes to be merged if both nodes have output and input reference values. Hence, the β_{Deg1} value as a reference would then be the {XOR (of all the MSBs in the node that β_{Deg1} comes from), min (of all the magnitudes in the node that β_{Deg1} comes from)}, i.e. β_{Deg1} node's 'output reference', and would act as the 'reference input' value to the degree-6 node. The degree-6 node would also need to provide a 'reference out' in the same manner to be the β_{Deg1}'s node 'input reference' for the nodes to be fully merge-able. Assuming that two nodes both have reference inputs and outputs they may be merged as shown below in Figure 3.6; the OP block is simply {XOR (two inputs' MSBs), min (two inputs' magnitudes)}. Note how OP blocks are used as to have the merged-node be merge-able itself. This is done as to have the new 'macro' node design support recursive nesting; i.e. each of the two internal blocks could be drawn as the block in where they reside and this pattern can repeat.

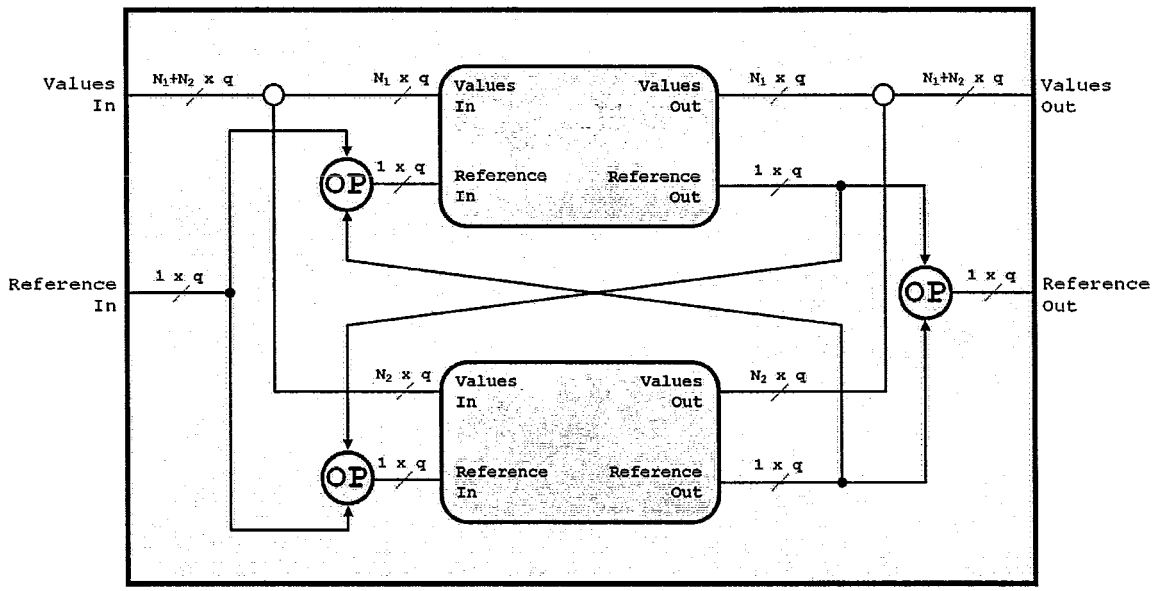


Figure 3.6: A depiction of how nodes of degree N_1 and N_2 can be merged to produce a merge-able node of degree $N_1 + N_2$.

For the case of merging 3 nodes, each with reference inputs and outputs is a more complex case and is shown below in Figure 3.7.

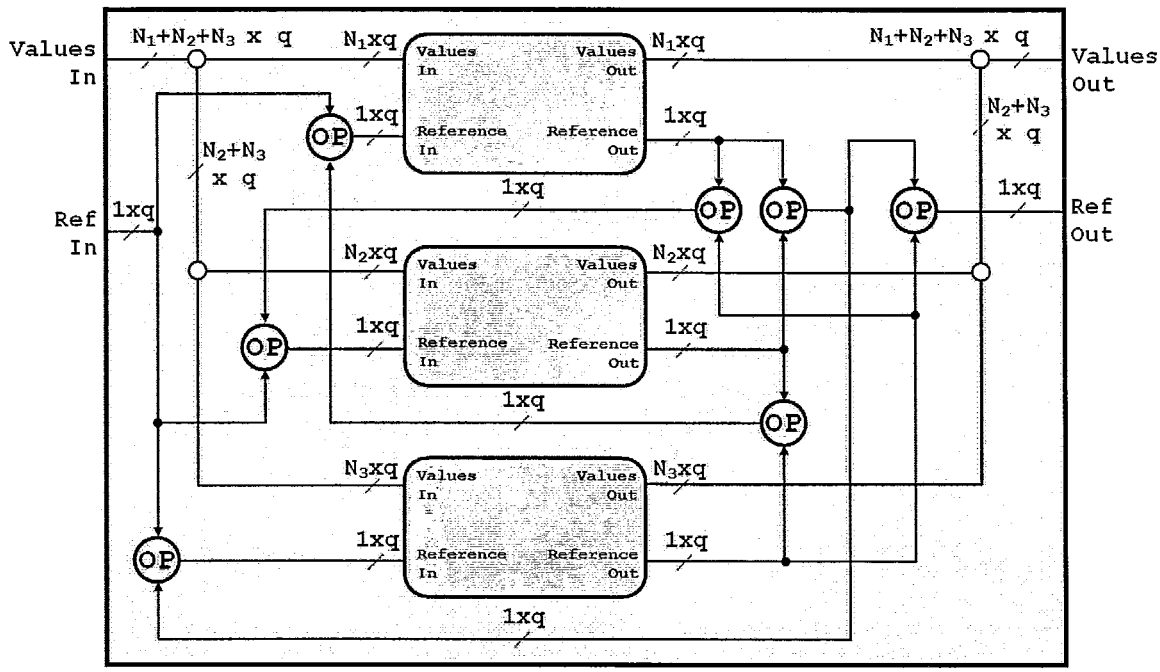


Figure 3.7: A depiction of how nodes of degree N_1, N_2, and N_3 can be merged to produce a merge-able node of degree $N_1 + N_2 + N_3$.

Providing these reference input and outputs can incur unnecessary additional cost if not done consciously. For example, if we wish to provide support for a reference input (recall the MSB and magnitudes are calculated independently, see Equation 3.1), each of the outputs can be updated with the reference input just before they are outputted as shown below in Figure 3.8.

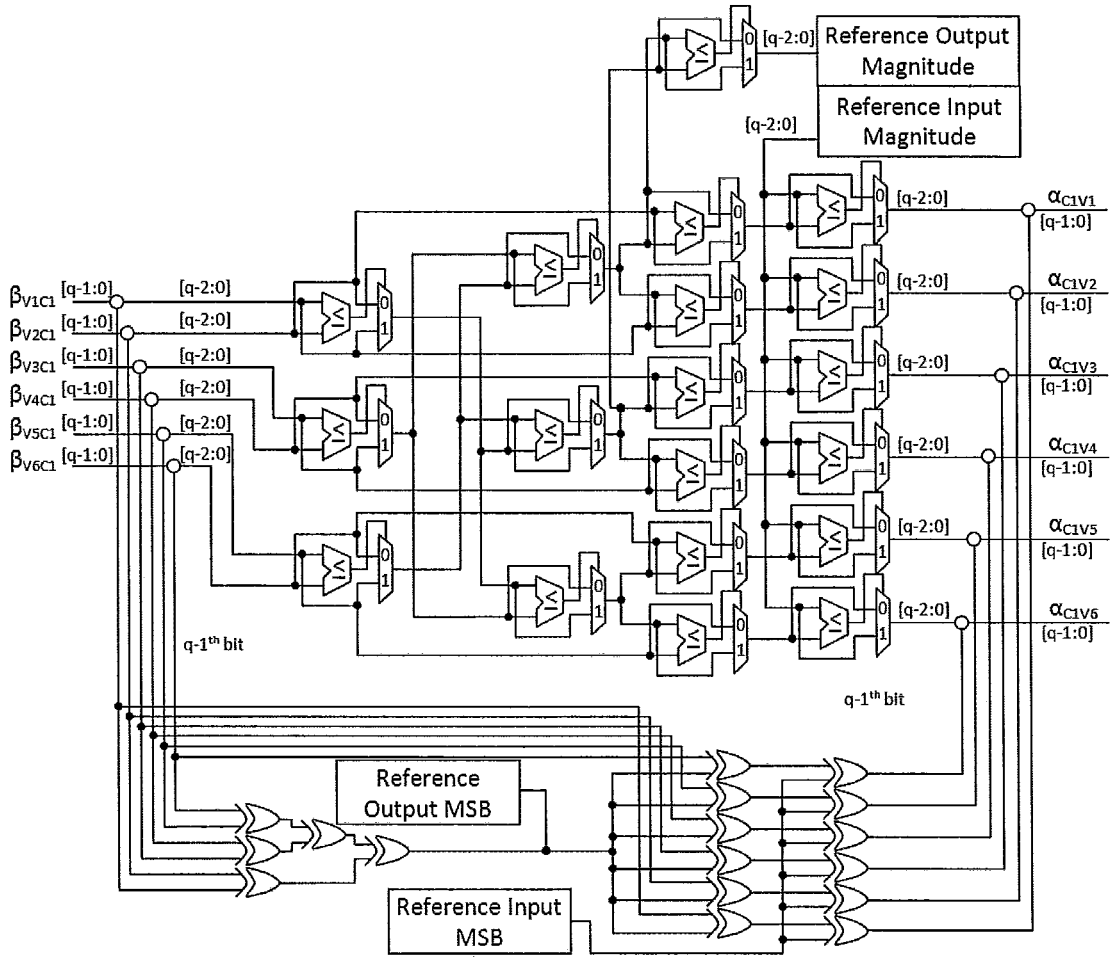


Figure 3.8: An inefficient inclusion of the Reference Input for the degree-6 example in Equation 3.1.

This is an inefficient approach as there is another approach which requires less additional minimum units (i.e. the repeating sub-block with the multiplexer and $\leq$ unit) and XOR units for inclusion of the reference input.

To include the reference output, one needs only to tap the correct wires and add an additional minimum unit; the cost for the MSB of the reference out is free. The more efficient inclusion of the reference input, with the same means of inclusion of the reference output, is shown below in Figure 3.9.

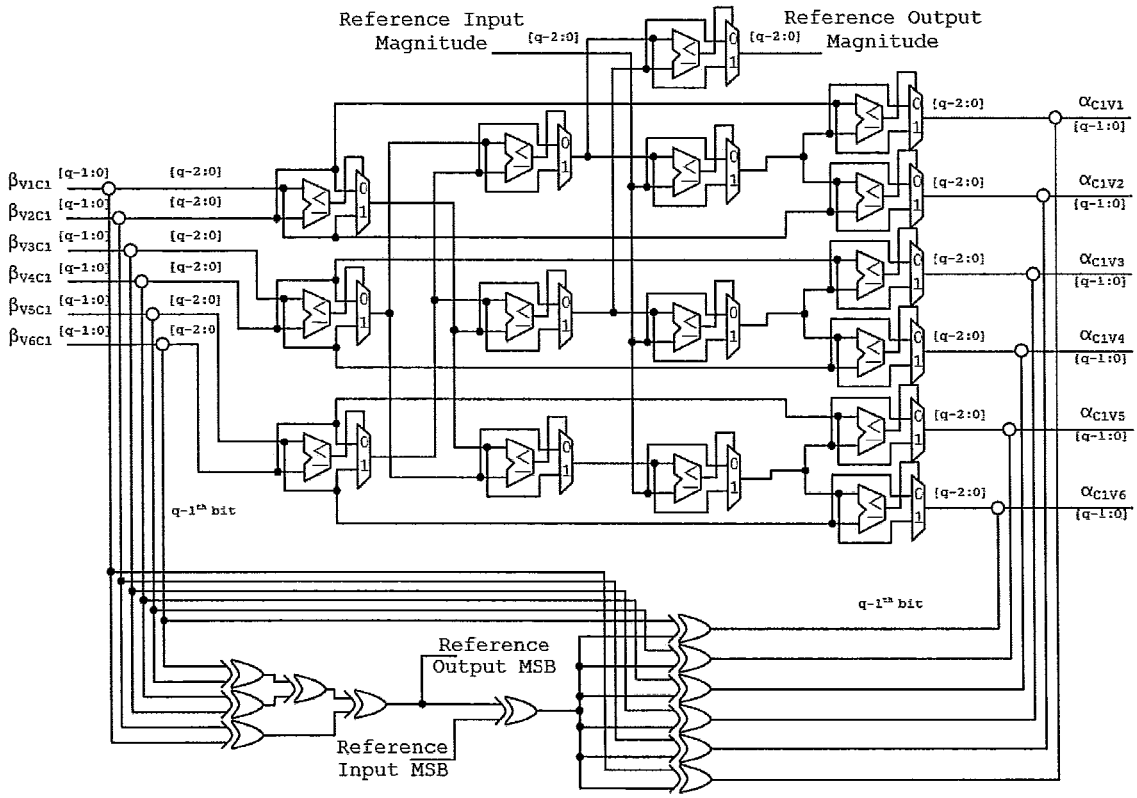


Figure 3.9: A new architecture that will support both a reference input and a reference output (architecture is modified from Figure 3.1).

Apparently, there is a cost incurred whilst adding the support for these reference inputs and outputs and such is depicted in Table 3.1 which conforms to the arrangement in Figure 3.4. It should be noted that the cost gains are relative

to the simple implementation solution. To elaborate, say that one needed to support one degree-29, two degree-14 and four degree-5 nodes. Referring to Table 3.1 costs of nodes without the overhead of the reference inputs/outputs may be determined (i.e. columns 'Cost of XORs' and 'Cost of Mins', where a single standalone node of degree-29 costs 57 XORs and 88 minimum units, or a single degree-14 costs 27 XORs and 39 minimum units). To have the 'simple implementation' would be the architecture wherein all the needed nodes exist in parallel, i.e. to have a degree-29 node, two degree-14 nodes, and four degree-5 nodes is a cost of 1x57 + 2x27 + 4*9 = 147 XORs and 1x88 + 2x39 + 4x9=202 minimum units. Alternatively, by using the reconfigurable nature of this work's nodes, a single degree-29 macro-block would be needed and would only cost 73 XORs and 87 minimum units; a savings of 50% XOR count and 57% minimum unit count. The details for all the savings are in Table 3.1, note that savings are not shown for the sub-blocks as they are not the macro blocks and are only building blocks.

Degree	Cost of XORs	Cost of XORs with Ref Alone	Cost of XORs with Ref Built	Savings %	Cost of Mins	Cost of Mins with Ref Alone	Cost of Mins with Ref Built	Savings %
4	7	8	-	-	6	9	-	-
5	9	10	-	-	9	13	-	-
6	11	12	-	-	12	16	-	-
7	13	14	-	-	15	20	-	-
8	15	16	-	-	18	23	-	-
13	25	26	(12+12)+7=31	-	36	44	(16+16)+7=39	-
14	27	28	(10+10+8)+7=35	-	39	47	(13+13+9)+7=42	-
15	29	30	(14+14)+7=35	-	43	52	(20+20)+7=47	-
16	31	32	(16+16)+3=35	-	46	55	(23+23)+3=49	-
29	57	58	35+35+3=73	50	88	104	42+42+3=87	57
30	59	60	35+35=70	59	91	107	47+47=94	60
31	61	62	31+31+10+3=65	52	95	112	39+39+13+3=94	56
32	63	64	35+35=70	62	98	115	49+49=98	63

Table 3.1: The relative costs for implementing the support for reference inputs and outputs as to have merge-able nodes.

3.4 – Implementation

When engineering a complex design, one must recognise that an organized approach to such a large problem is essential. In accordance with this we put forth the following flow chart of the chosen design process shown below in Figure 3.10.

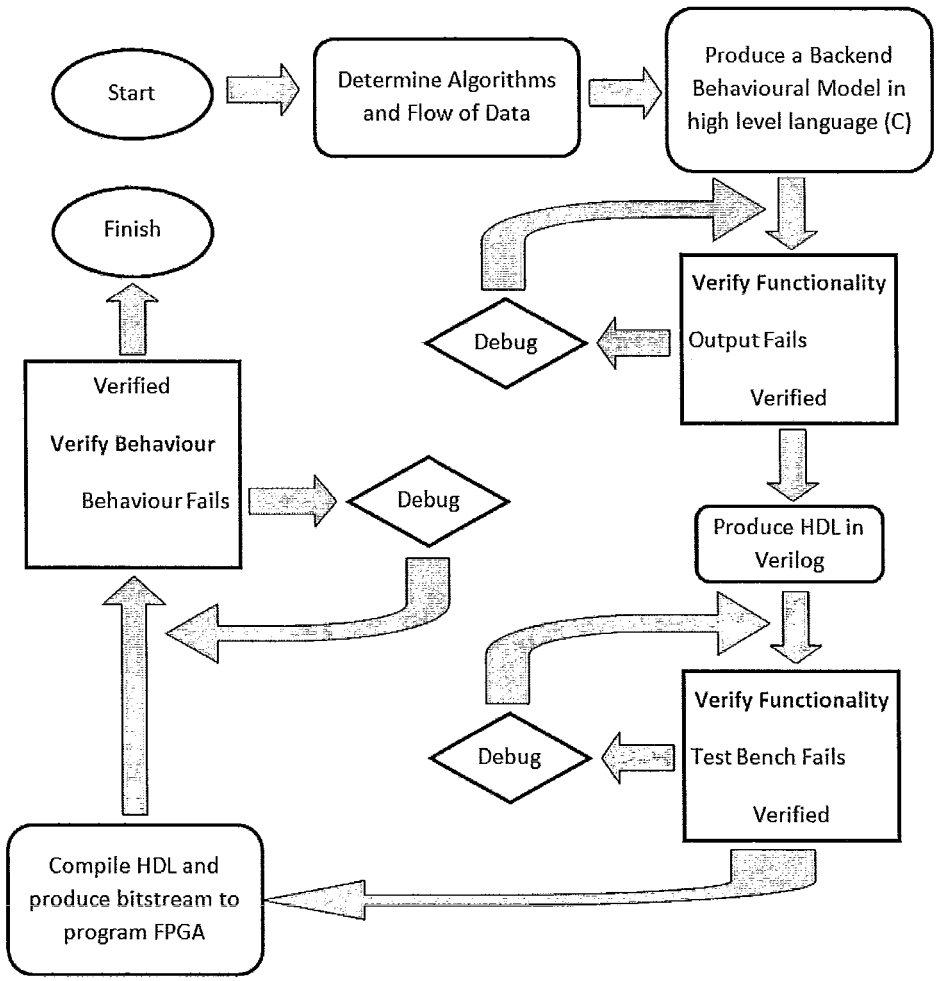


Figure 3.10: Flowchart for design process utilized by this thesis.

To outline the steps:

- Determine algorithm and flow of data
    - The algorithm needs to be established. It is determined by researching what is currently in the public domain; algorithm decision is based on what could be the most hardware efficient solution. Final decision of algorithm was with UMP.
    - The needed flow of data, here the message passing structure, is a good indication of needed complexity for the datapath; fewer complexes are beneficial.
- Produce a backend behavioural model in a high level language; C programming language as in [19]
    - The reason why a high level language such as C is used is to alleviate the process of debugging and coding; runtime is not of major concern as long as runtime is on the order of seconds to minutes.
    - The backend must also output the necessary data files for the test bench which runs later in the design flow.

- Verify functionality
    - Simply ensure that the decoding is successful, and if not, then debug until the verification passes; use output files to determine bugs in the code.
- Produce Hardware Description Language (HDL) at Register Transfer Level (RTL); Verilog HDL as in [45]
    - Translate as much C code as possible into Verilog, exploiting the similarity in dataflow. The advantage of a behaviour model's similarity in dataflow when compared to that of the hardware implementation is appreciated in this step.
    - Produce the remaining needed modules along with a top level organization of the modules.
- Verify functionality
    - Utilize a test bench (written specifically for this design or if time permits have test bench generated automatically from the backend) to verify functionality; upon failure, debug until failure does not occur. The test bench must test enough cases as to correctly verify functionality. Lucky cases must not be accepted as proof of functionality.

- Compile HDL and produce bitstream to program FPGA
    - Now that the design is verified, the HDL is compiled as to create a bitstream that will program the FPGA so the following step may be performed.
- Verify behaviour
    - The complete and correct behaviour of the design is tested. This will verify if the FPGA implementation operates correctly.
    - If the behaviour is incorrect in any way, observation tools may be utilized to probe internal signals in the FPGA-implemented design as to determine the inconsistencies from that of the test bench.
- Finish
    - If all is functional, design is complete to the scope of this thesis.

3.4.1 - Behavioural Model

To be able to verify the operation and functionality of the final design, and even to be able to debug the first draft of the design, a software-based behavioural model is required; already depicted in the design flow of this work in Figure 3.10.

There are two main approaches that can be used as to produce the software behavioural model: a pure software model may be derived as to simply compute the necessary calculations and operations with coding optimized for an instruction machine (i.e. the computer running the code), or a 'closer to hardware' model may be derived that also computes the necessary calculations and operations, yet with behaviour (i.e. how the data is organized and moved around) closer to that of the final hardware implementation.

The advantage of the earlier approach is that the code will run fast and the conceptual portion of the design can be proven. Yet such an approach does not yield as closely matching debug data (for the hardware debugging) as does the later approach, which is so closely related to the hardware design to be debugged. Captured output of the software backend behavioural model is shown below in Figure 3.11 as demonstration of the backend program flow.

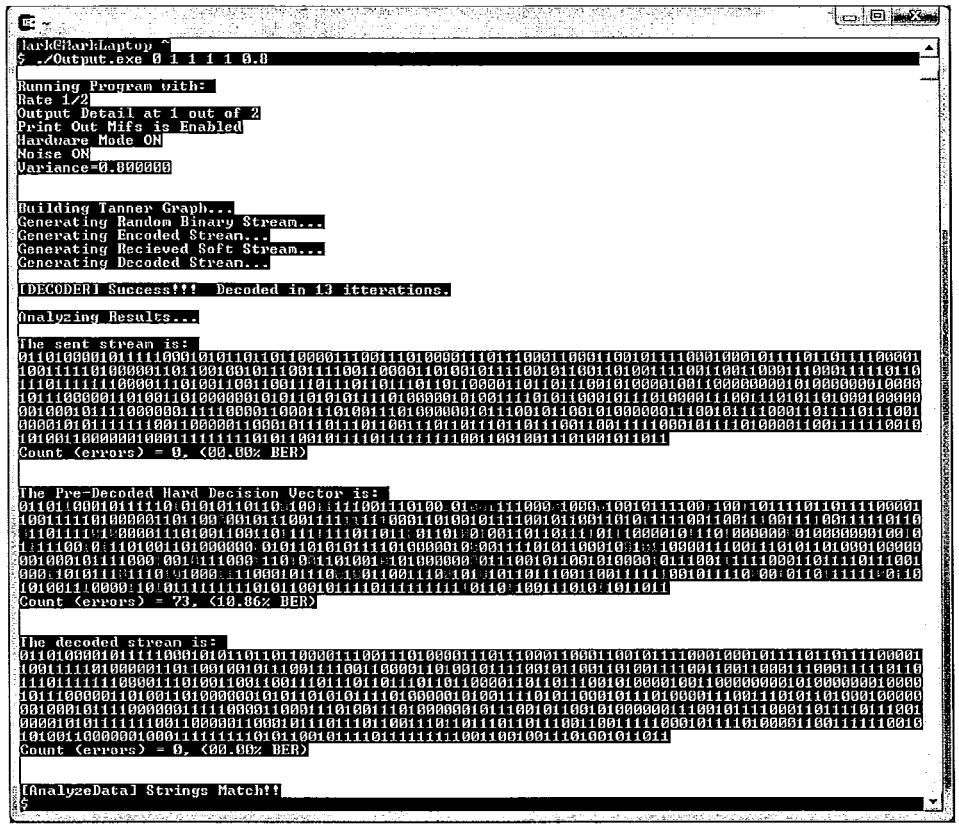


Figure 3.11: Captured output from cygwin running the software backend behavioural model for a single instance.

3.4.2 - Generic VLSI Architecture

There are many possible ways to utilize this work's contribution in the chosen case study as a proof of concept. The generic architecture for the decoder is shown below in Figure 3.12.

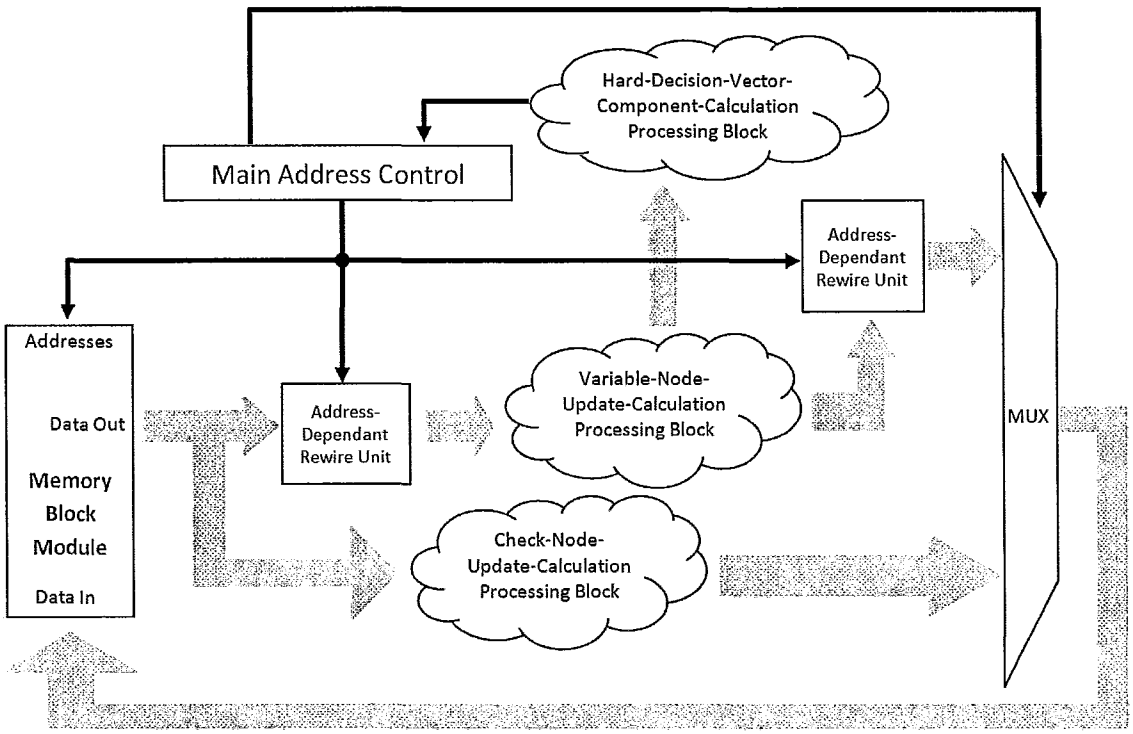


Figure 3.12: The general LDPC decoder architecture for implementation in the proof of concept.

The parameters driving the variety of possible resultant constructions of the above generic architecture mostly manifest in the number of Variable and Check Node-Update-Calculation engines in each Processing Block, along with the Memory Block Module. The other sub modules also experience alterations as the different schemes are explored, yet these differences are moot when compared to the needed alterations for the earlier stated blocks. Let us clarify some of the sub modules that have not yet been discussed in detail.

The Main Address Control Unit is the centre of the datapath control in this device. It will control all the addressing and muxing in the entire decoder and will also control the evaluation of the Hard-Decision-Vector-Component-Calculation.

The Hard-Decision-Vector-Component-Calculation Processing Block simply takes in the individual results from the Hard-Decision-Vector-Component-Calculation in the Variable-Node-Update-Calculation Processing Block, compiles them into a single word, and tests residue; the result is reported back to the Main Address Control Unit.

The Address-Dependant Rewire Units are particular to this work's approach to decoding and are resultant from both how the matrix to be supported is constructed along with how the memory is organized. In particular, a set of q 2:1 MUXs is needed for every value that is not translated between matrices and a maximum of q 2:1 MUXs is needed in addition for each input to the Variable-Node-Update-Calculation Processing Block for some schemes. A worst case scenario architecture wherein L different inputs are rewired to L different outputs is provided below in Figure 3.13. This is assuming three as the max number of possible different inputs (selections from the bus of all inputs) selected. Note that not every output need be a selection of one of three inputs. In fact, the number of different inputs that one output may be is directly related to the chosen configuration of the general architecture in Figure 3.12 in addition to being related

to the parity check matrices implemented by the decoder. The number of inputs can vary from 2 to as many as 5 as design configurations are explored; architecture configurations are elaborated in section 3.4.3.

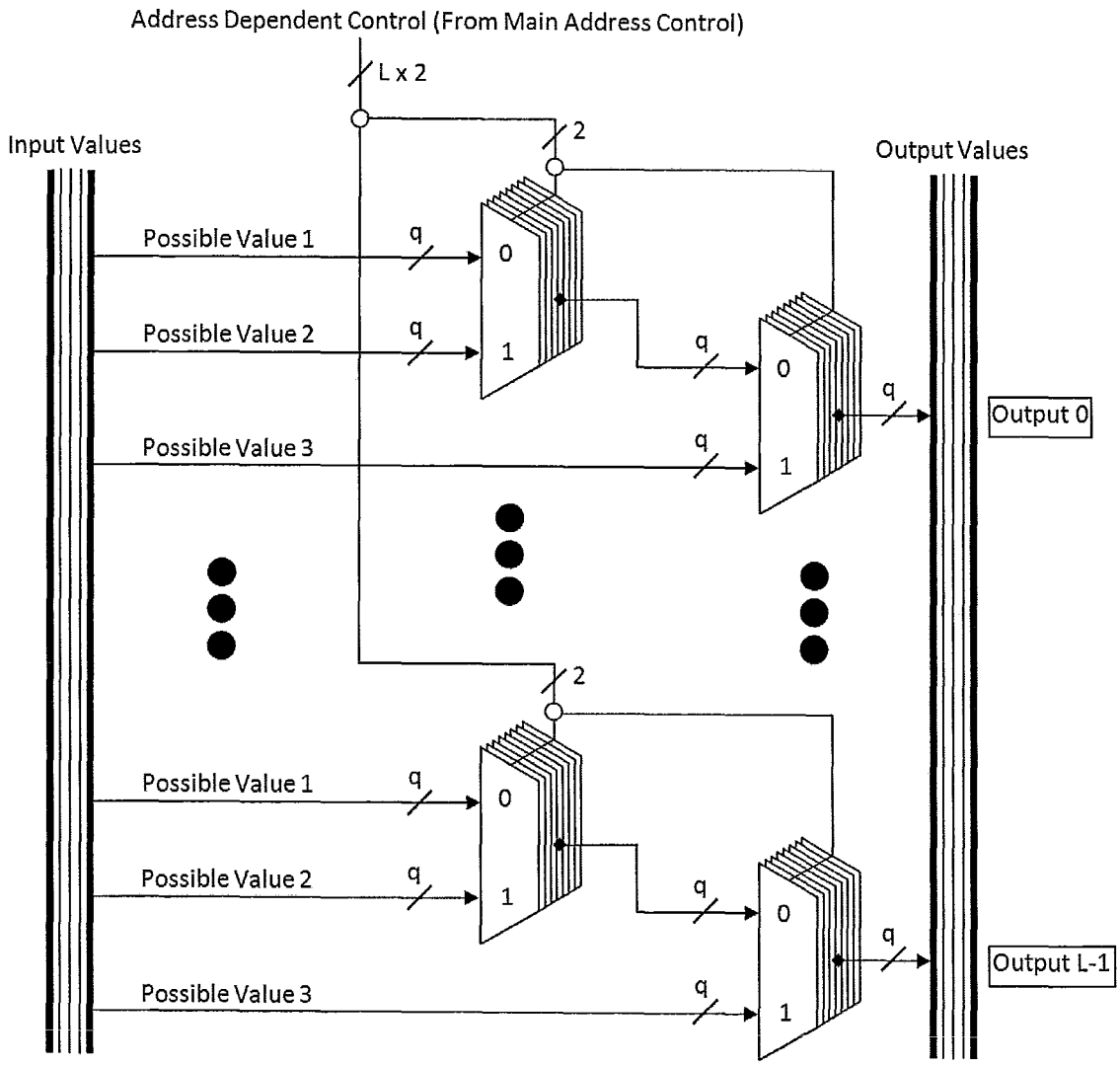


Figure 3.13: Architecture of worst case scenario Address-Dependant Rewire Unit.

The Memory Block Module is the wrapped module containing all the storage for the messages that are passed. Internal organization is simply a collection of parallel smaller memories as to meet the large bandwidth requirement, along with some simple internal data dropping via the individual memory write enables (necessary for the data dropped through 'pseudo puncturing', see section 3.2). An architecture for this memory stacking as to boost bandwidth is provided below in Figure 3.14.

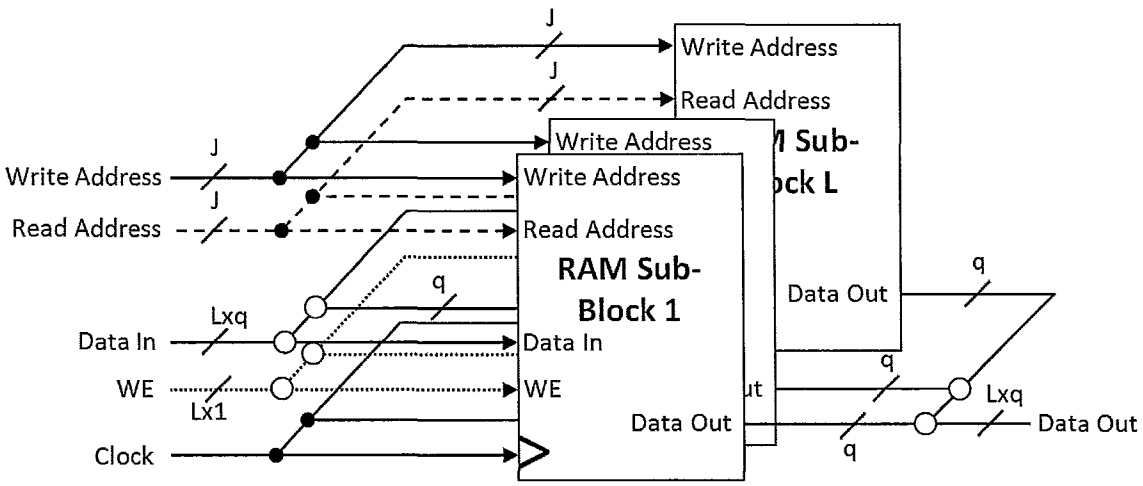


Figure 3.14: An architecture of the Memory Block Module.

It should be noted that the above figure is assuming a bandwidth of L input words of length q to be written and L output words of length q to be read each clock cycle; L = 366 in the final implementation and is discussed in section 3.4.3. The variable J is resultant from the number of addresses in the sub blocks, i.e. the

number of locations to where data of width q may be written, and is equal to ceiling($\log_2$(number of addressable memory locations in one RAM Sub-Block)). Hence for 7 addresses (the chosen organization for this thesis' implementation), J=3.

The Variable-Node-Update-Calculation Processing Block is the processing block that which contains all the parallel instances of the variable node processing blocks; the quantity is a design constraint and is addressed in the following section 3.4.3.

The Check-Node-Update-Calculation Processing Block is the processing block that which contains all the parallel instances of the macro check node processing blocks; the quantity is a design constraint and is addressed in the following section 3.4.3.

The general architecture in Figure 3.12 is too general as to use to directly create the final design; the level of parallelism still needs to be addressed and is done in the following section 3.4.3.

3.4.3 - Final Implementation and Results

When implementing this thesis' contribution in the case study (IEEE 802.15.3c LDPC Rate 1/2, 3/4, and 7/8 matrices), there is still some design to address concerning the parallelism to be applied to the Check-Node-Update-Calculation and Variable-Node-Update-Calculation Processing Blocks. To achieve the needed throughput as detailed by the case study of this thesis, i.e. 802.15.3c, a table was produced as to determine the best configuration via internal relationship; Table 3.2 shows this below

Comb Block Count	Memory Organization	#cc / Iteration	Est. Throughput With Clock @ 270MHz 8-Iter (1E6 b/s)	2:1 MUX Total	2 Input LUT Total	Relative 2:1 MUX to Comb1	Relative 2 Input LUT to Comb1	Relative Throughput	Relative Cost to Throughput Gain
1	21Addr x (29+30+31+32)	44	515	9868	54511	1	1	1	1
3	7Addr x (29+30+31+32) x 3	14	1620	27416	157407	2.7783	2.8876	3.1456	0.90
7	3Addr x (29+30+31+32) x 7	8	2835	61438	363199	6.2260	6.6629	5.5049	1.17
21	1Addr x (29+30+31+32) x 21	4	5670	180100	1083471	18.251	19.876	11.01	1.73

Table 3.2: A depiction of the cost analysis used for design.

To explain Table 3.2, there are 4 possible arrangements that can be chosen for the levels of parallelism applied to the general architecture in Figure 3.12. The first option would use one block of the variable node / check node processing engines in the Variable and Check Node-Update-Calculation Processing Blocks; hence the comb block count equalling one. What is meant by one 'block' is one set of the four (degree-29, degree-30, degree-31, and degree-32) check-node-macro-blocks described in section 3.2, along with the 32 variable-node-update-calculation blocks needed to have the bandwidth line up (i.e. if as much as 29+30+31+32=122 values need to be processed, as many as 32 4-input variable-node-update-calculation blocks are needed).

The elaboration of the check-node-update-calculation block of a single block (comb-1) and a three block (comb-3) are illustrated below in Figure 3.15 and Figure 3.16 respectively. Note that the comb blocks are the check node and variable node blocks and as a result the comb block has one set of outputs for check updates and one set of outputs for variable node updates; see Figure 3.12 and note the large MUX. Also, the Rewire block is an abbreviation of the Address-Dependant Rewire Unit.

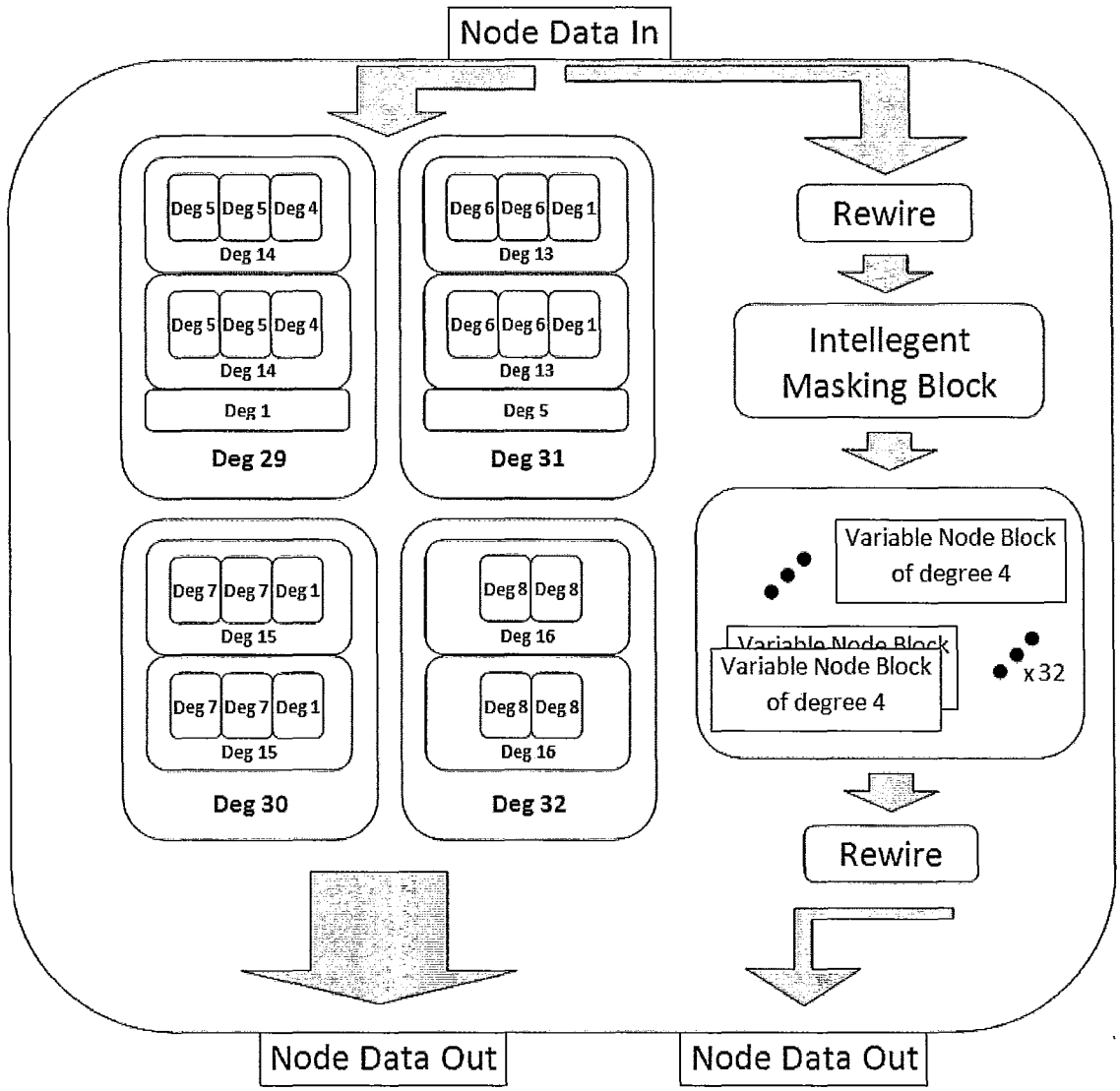


Figure 3.15: An illustration of a comb-1 block.

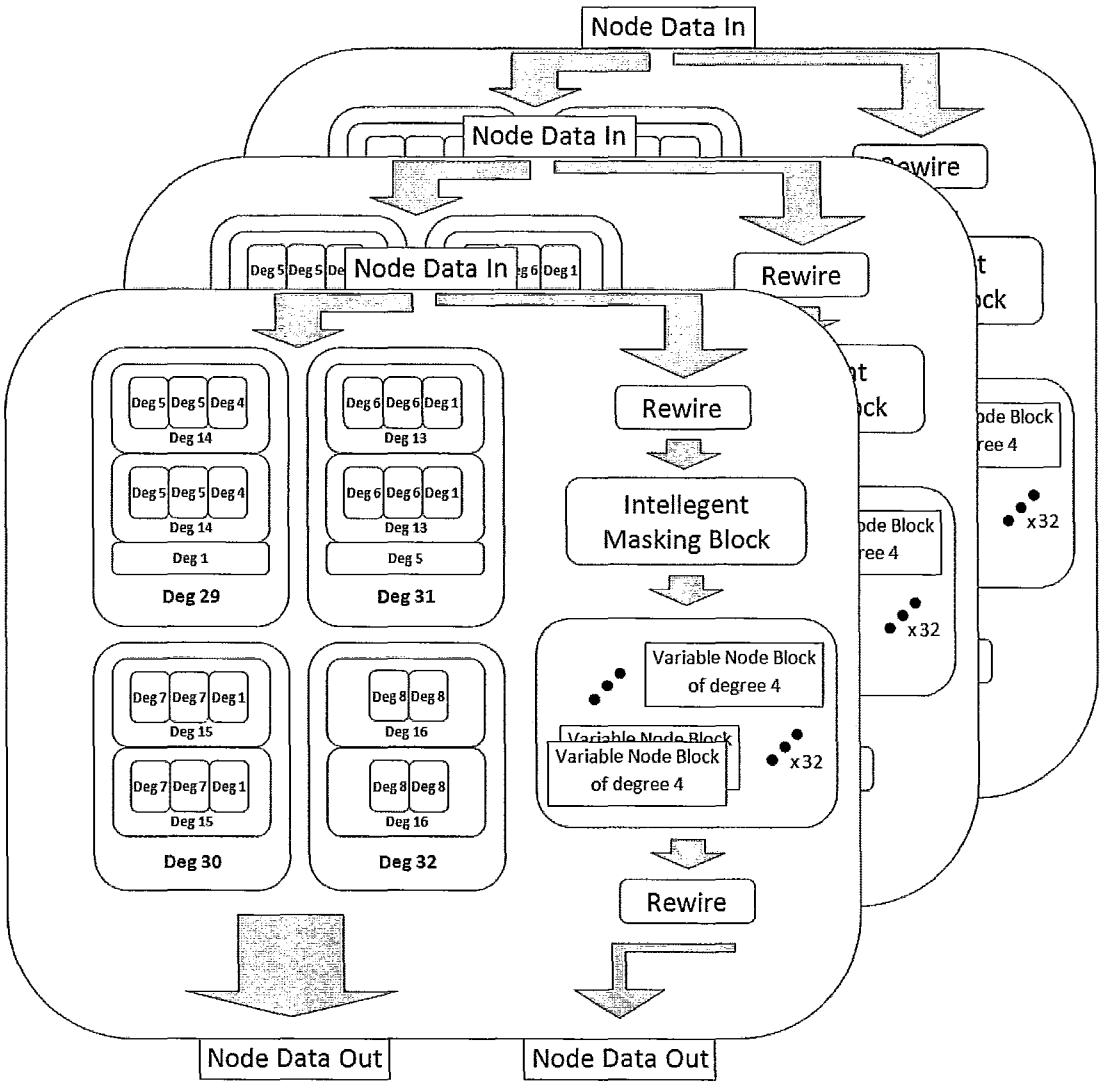


Figure 3.16: An Illustration of a comb-3 block.

To cover all of the needed space for all the traversing messages we need a memory organization that has 21 different addresses for 29+30+31+32 different instantiations (each memory would have a bitwidth equal to q in Figure 3.9, which

equals the chosen bitwidth for the decoder which is eight in this design). This is because the case study's design parameter K (see Figure 1.10) is equal to 21.

The throughput for this choice is dependant on the number of iterations. Because the number of iterations for the variable and check node updates are equal, along with the 21 addresses to process + 1 clock cycle (cc) for latency, 44cc are required to process 672 bits (the codeword block length). A throughput of 672 bits / 44cc * estimated clock of 270MHz (the estimated clock is assumed to be 10x the fastest allowable clock for the FPGA implementation) or simply 515 Mb/s is experienced with the costs of 9868 2:1 MUXs / 54511 LUTs.

Although the performance gap between ASICs and FPGAs has been improved down to 3.5x over the past few years, as was accepted in IEEE Transactions on Computer Aided Design in 2006, because our delays vary due to routing and random logic (i.e. comparators, XOR gates, muxes) in the FPGA we choose the FPGA/ASIC ratio, in terms of clock period, to be 10x.

It can be seen that the first design which satisfies the throughput constraint of 1 – 2 Gb/s, comb-3, also has a relative gain over the first design option, comb-1; a useful fact due to the more parallel designs' (i.e. comb-7 and comb-21) low chance of fitting on the chosen FPGA and the less parallel design having insufficient throughput.

When implementing the second design option along with all the needed peripherals, the following resource usage is needed on the Cyclone II EP2C70F896C6 FPGA (compiled with Quartus II 7.2 Student Edition): 1 PLL; 47,080 Logic Elements; 693 Registers; and 29,568 bits of memory. The actual memory needed is only 20,496 bits and results in a memory bandwidth of (29+30+31+32) * 3 * 8bits = 2,928 bits/cc (i.e. 122 * 3 = 366 different 7 address memory blocks). Note that the usage was determined on Quartus II V7.2-203 SP2 and that the critical path delay is 35.792 ns experienced on the path from: the Memory Block Module through the Check-Node-Update-Calculation Processing Block and back through the multiplexer to the Memory Block Module. This path is illustrated as it traverses through the general architecture in Figure 3.12 below in Figure 3.17.

The design was also implemented on a Stratix II EP2S180F1508C5 as to be able to compare the area and critical path of the chosen implementation scheme vs. the basic parallel implementation. As a result the chosen implementation has a resource usage of: 36,162 Adaptive LUTs (ALUTs), 700 registers, and 29,568 bits of memory; a register to register critical path of 34.55 ns (28.94 MHz), through the same path as in Figure 3.17, is experienced. The basic parallel implementation has a resource usage of: 48,093 ALUTs, 702 registers, and 29,568 bits of memory; a register to register critical path of 33.51

ns (29.85 MHz), through the same path as in Figure 3.17, is experienced. Recall the only difference between these two implementations is that the Check-Node-Update-Calculation Block developed by this thesis is replaced with a functionally equivalent block wherein the nodes of various degrees are simply instantiated in parallel. The reason why another device was used was due to the fact that the parallel version did not fit on the Cyclone II device. The Stratix II device was used as to be able to compare the two approaches properly. These numbers suggest that for a cost of (1 – 28.94/29.85) = 3% reduction in throughput (recall the bandwidth is the same in terms of cc so clock rate directly relates to throughput), an area savings of (1 – 36162/48093) = 24.8% is experienced. Please note that these numbers are relative to themselves and as a result the saving percentages cannot be guaranteed, they are however fairly indicative of the expected savings.

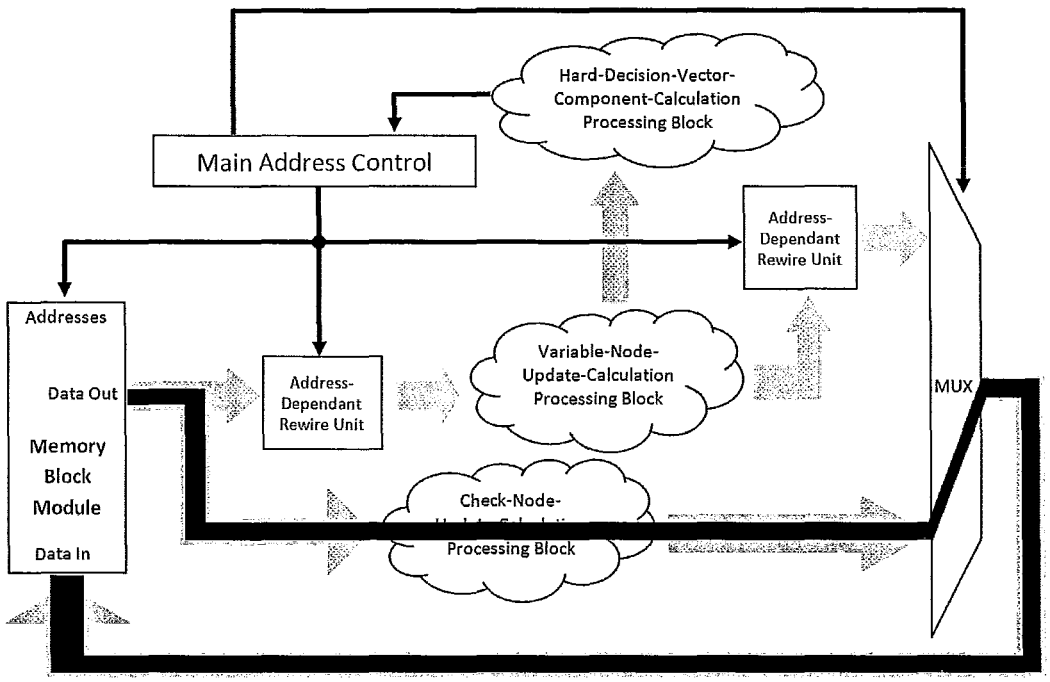


Figure 3.17: Illustration of the critical path through the general architecture.

The reason why the critical path traverses through this path is clear when recalling the combinational complexity incurred in the Check-Node-Update-Calculation Processing Block as a result of the nested node design. What were once simple connections between input and output (i.e. in a single base node, there is no reference input, so the output can be directly determined by the inputs alone) are now weaved chains of dependency. In other words, regarding Figure 3.6, the output of the degree N_2 node is dependant on the degree N_2 input and reference input. The reference input of the degree N_2 node is dependent on the inputs to the degree N_1 node. Hence, the weaved chain expression earlier said is

derived from the some-what 'zigzag' paths that data may traverse with one such path being: Inputs → degree N_1 inputs → degree N_1 reference output → degree N_2 reference input → degree N_2 output. As levels of nesting increase this 'zigzag' path increases in length resulting in an increase in the register to register critical path length. A detailed illustration of the degree-31 macro block (see section 3.2) is provided below in Figure 3.18 with a possible critical path highlighted (due to some internal symmetry this is subject to change as the final routing inside the FPGA will truly determine the critical path).

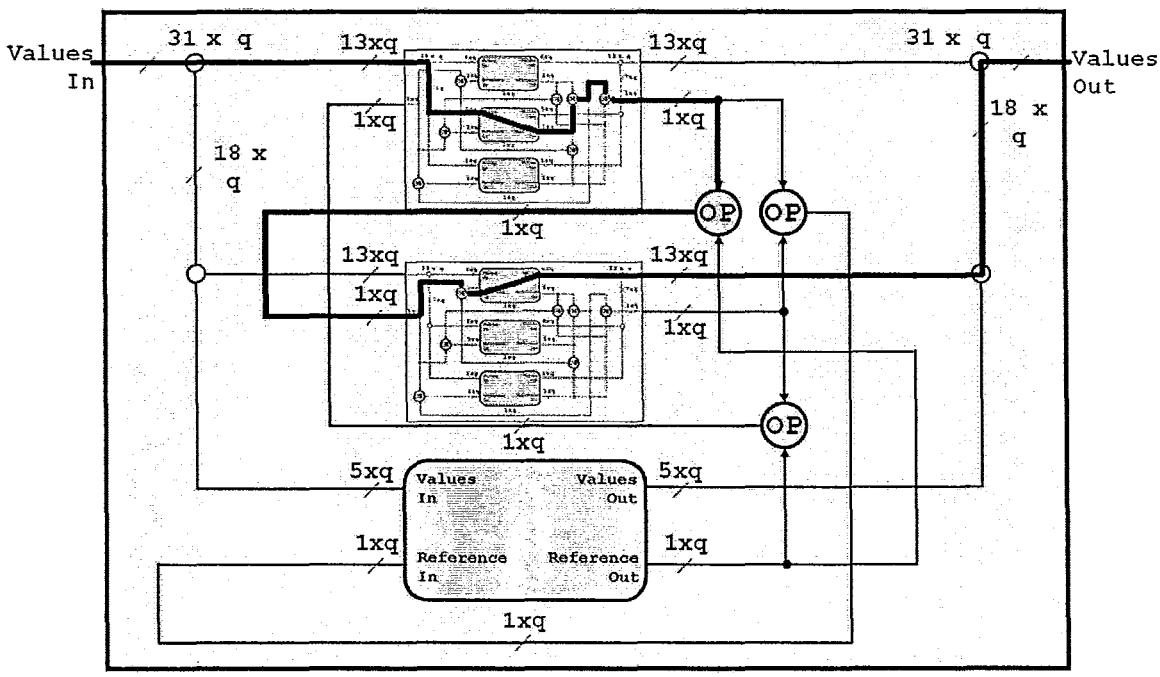


Figure 3.18: Illustration of a possible critical path through the degree-31 macro node from section 3.2.

It should be noted that in the implemented design, in the Cyclone II device, the 29,568 bits of memory are needed only due to the limitations of the FPGA utilized. To continue, 7 addresses * (29+30+31+32) * 3 * 8 bits is only 20,496 bits of memory, but because not all memory address/width configurations are possible (the limitation of the FPGA) there is some wasted space. This wastage could be avoided in an ASIC implementation as the standard library could be much larger and could contain more flexible memory configurations.

Chapter 4

4 – Conclusion

In this thesis we have discussed the application of nodal exploitation in the context of low density parity check, LDPC, codes. Chapter 1 provided the necessary foundation for understanding the basics of LDPC codes, along with the encoding, decoding, and construction of LDPC codes. Chapter 2 analyzed the current architectures in the public domain and related these designs via a consistent comparison table created by [3], in addition to providing a detailed analysis of three state of the art architectures design attributes. Chapter 3 explained the details involved in decoding LDPC codes and provided architectures and a general methodology for building nodal processing blocks in which a significant relative savings of area is experienced when compared to the alternative of the parallel instantiation of nodes. This utilizes the inherent nesting of nodes in the parity check matrices resultant from the matrices' construction method (Chapter 1.2.4). Also provided by Chapter 3 is the design methodology utilized by this thesis and the results of the final implementation of the LDPC decoder for multi-rate support of the parity check matrices in this thesis' case study, namely 802.15-3c.

4.1 - Advantages

As more wireless devices are built, more advanced coding schemes are desirable as to fully benefit from the advancements in said devices. One aspect of the wireless coding schemes is that of multi-rate support; channel parameters can change greatly if the wireless application is to be utilized in short range (i.e. people can walk past operating devices and greatly change the channel parameters forcing an alternative rate to be used as to be able to get the best code performance for the new channel). Accepting that multi-rate support is a necessity this thesis' work can greatly contribute to the design of LDPC decoders.

This work has potential to reduce the needed area for a LDPC decoder that needs to operate at multiple rates. The matrix construction necessary for this work is simple and commonly accepted in standards such as WPAN. Thus this thesis' work may be applied to a myriad of LDPC parity check matrices used in industry and may have the potential to create a new branch of code design which even further facilitates nodal nesting structure exploitation. Also, as described by section 4.3, this work may be applied to a family of algorithms in which the desired form is present; see section 4.3 for more details.

4.2 - Limitations

The work in this thesis does prove helpful in the scope of multiple rate support. Yet if only a single rate is desired, the exploitations utilized by this work are moot and thusly this work does not apply to such a situation. Alternatively if the number of rates to be supported by the decoder comprises a long list, the nodal exploitation has only been verified to support the rates of the form (R-1)/R where R$\in$ $|\mathbb{Z}|$and R>1. The additional complexity for supporting other rate forms could negatively impact the area gains for the rates that are of a different form (i.e. 5/7 is not of the form (R-1)/R, thus to have a macro node support this rate likely requires more complexity). In other words if matrices of varying rate forms need be supported the macro nodes may have a more complex structure. To support alternative rates could possibly be done by having some values propagate to two locations and other values propagate to a different integer number of locations. This detail was not explored as the case study had matrices of compatible rate form. This additional complexity is not required if all the design parameters can be controlled (i.e. matrix construction), for then we are able to design the code foundation, or set of matrices to be supported, as to even improve the final performance.

4.3 - Future Work

There is a factor about this work which has not yet been addressed. These extensions to this work are elaborated below.

Concerning the formulae for the node update calculations and recalling that the key attribute was the fact that each output would only be a function of all the inputs except said output's counterpart, this work's applicability may be extended to any algorithm of said nature wherein different degrees need support. This is assuming that the formula or formulae to be processed are symmetric, i.e. F(A,B,C) = F(B,C,A), and separable, i.e. F(A,B,C) = F(A) ∩ F(B) ∩ F(C) such as addition, the max function, or the min function. Regarding Figure 3.6 allow the 'OP' calculation blocks to be the operation performed by the algorithm which we desire to implement and allow the sub-blocks to be built in a similar pattern as they are in the macro-block.

Also, with intelligent node grouping as to minimize interconnection complexity, the same hardware may be used to process a collection of grouped nodes separately or as a single node via the reference values regardless of internal computation (assuming the algorithm is of the earlier assumed form). In future work the possible applications would be explored in other possible fields, such as hardware acceleration and information theory.

Bibliography

[1] Daisuke Abematsu, Tomoaki Ohtsuki, Sigit PW Jarot, Tsuyoshi Kashima. Size Compatible (SC)-Array LDPC Codes. *IEEE VTC-2007*, pp. 1147-1151.

[2] C.H. (Kees) van Berkel. Multi-Core for Mobile Phones. *IEEE Proc. DATE 2009*, pp. 262-267

[3] Torben Brack, Frank Kienle, Norbert When. Disclosing the LDPC Code Decoder Design Space. *IEEE Proc. DATE 2006*, pp. 200-206

[4] T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, N. Wehn, N.E. L'Insalata, F. Rossi, M. Rovini, L. Fanucci. Low Complexity LDPC Code Decoders for Next Generation Standards. *IEEE Proc. DATE 2007*, pp. 331-336

[5] D. Brock, G. Moore. *Understanding Moore's Law Four Decades of Innovation*. Online Publication, Chemical Heritage Foundation, 2006

[6] Zhiqiang Cui, Zhongfeng Wang, Youjian (Eugene) Liu. High-Throughput Layered LDPC Decoding Architecture. *IEEE Trans. VLSI 2009*, VOL. 17, NO. 4, pp. 582-587

[7] John Dielissen, Andries Hekstra, Vincent Berg. Low Cost LDPC Decoder for DVB-S2. *IEEE DATE 2006*, pp. 1-6

[8] European Telecommunications Standards Institude (ETSI). *Digital Video Broadcasting (DVB) Second generation framing structure for broadband satellite applications; EN 302 307 V1.1.1*. www.dvb.org.

[9] M. Fossorier. Quasi-Cyclic Low-Density Parity-Check Codes From Circulant Permutation Matrices. *IEEE Trans. Inf. Theory 2004*, vol. 50, pp. 1788-1793

[10] M. Franceschini, G. Ferrari, and R. Raheli. Does the performance of LDPC codes depend on the channel?. *IEEE Trans. Communication 2006*, pp. 2129-2132

[11] R. G. Gallager. Low density parity-check codes. *IRE Trans. Inf. Theory 1962*, pp. 21-28

[12] F. Guilloud, E. Boutillon, and J.L. Danger. λ-Min Decoding Algorithm of Regular and Irregular LDPC Codes. *3nd International Symposium on Turbo Codes and Related Topics*, Brest, France, pp 451-454, Sept. 2003

[13] Kiran Gunnam, Gwan Choi, Weihuang Wang, Mark Yeary. Multi-Rate Layered Decoder Architecture for Block LDPC Codes of the IEEE 802.11n Wireless Standard. *IEEE ISCAS 2007*, pp. 1645-1648

[14] Kiran K. Gunnam, Gwan S. Choi, Mark B. Yeary, Mohammed Atiquzzaman. VLSI Architectures for Layered Decoding for Irregular LDPC Codes of WiMax. *IEEE ICC apos 2007*, pp. 4542-4547

[15] D. Hocevar. A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes. *IEEE SIPS 2004*, pp. 107-112

[16] IEEE, Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs): Amendment 2: Millimeter-wave based Alternative Physical Layer Extension IEEE P802.15.3c/D00-2003

[17] Marjan Karkooti, Joseph R. Cavallaro. Semi-Parallel Reconfigurable Architectures for Real-Time LDPC Decoding. *IEEE ITCC 2004*, pp. 571-585

[18] Marjan Karkooti, Predrag Radosavljevic, Joseph R. Cavallaro. Configurable LDPC Decoder Architectures for Regular and Irregular Codes. *Springer JSPS 2008*, pp. 73-78

[19] B. Kernighan, D. Ritchie. *C Programming Language*. Prentice Hall PTR, 2 ed., April 1988, ISBN-13: 978-0131103627

[20] F. Kienle, T. Brack, and N. When. A synthesizable IP Core for DVB-S2 LDPC Code Decoding. *IEEE DATE 2005*, pp. 100–105

[21] F. Kschischang, B. Frey, and H. Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Trans. Inf. Theory 2001*, vol. 47, pp. 498-519

[22] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong. Efficient encoding of low-density parity-check codes. *IEEE Trans. Communication 2006*, pp. 71-81

[23] S. Lin and D. J. Costello, Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 2nd edition, Upper Saddle River, NJ., 2004.

[24] Chih-Hao Liu, Shau-Wei Yen, Chih-Lung Chen, Hsie-Chia Chang, Chen-Yi Lee, Yar-Sun Hsu, Shyh-Jye Jou. An LDPC Decoder Chip Based on Self-Routing Network for IEEE 802.16e Applications. *IEEE JSSC 2008*, VOL. 43, NO. 3, pp. 684-694

[25] G. Liva, E. Paolini, and M. Chiani. Simple Reconfigurable Low-Density Parity-Check Codes. *IEEE Trans. Communication 2005*, vol. 9, pp. 258-260

[26] Jin Lu, Jos´e M. F. Moura. Linear Time Encoding of LDPC Codes. http://arxiv.org/abs/0810.2781, 2008

[27] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. *IEEE Trans. Inf. Theory 2001*, vol. 47, pp. 569-584

[28] M. Mansour and N. Shanbhag. High-Throughput LDPC Decoders. *IEEE Trans. VLSI 2003*, pp. 976–996

[29] M. Mansour and N. Shanbhag. A Novel Design Methodology for High-Performance Programmable Decoder Cores for AA-LDPC Codes. *Springer JVSP 2005*, vol. 40, pp. 371–382

[30] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994, ISBN: 0-07-113271-6

[31] Marghoob Mohiyuddin, Amit Prakash, Xiang Wu, Adnan Aziz. A Reconfigurable Fabric and Associated CAD Algorithms for Multirate LDPC Decoding. *39th Asilomar Conference SSC 2005*, pp. 718-722

[32] C. Kopp. *Moore's Law and its Implication for Information Warfare*. 2000

[33] A. Morello and V. Mignone. DVB-S2: The Second Generation Standard for Satellite Broad-Band Services. *IEEE Proc. 2006*, vol. 94, pp. 210–227

[34] Stefan Müller, Manuel Schreger, Marten Kabutz, Matthias Alles, Frank Kienle, Norbert When. A Novel LDPC Decoder for DVB-S2 IP. *IEEE DATE 2009*, pp. 1308-1313

[35] Akiyuki Nagashima, Yuta Imai, Nozomu Togawa, Masao Yanagisawa, Tatsuo Ohtsuki. Dynamically Reconfigurable Architecture for Multi-Rate Compatible Regular LDPC Decoding. *IEEE APCCAS 2008*, pp. 705-708

[36] Daesun Oh, Keshab K. Parhi. Low Complexity Decoder Architecture for Low-Density Parity-Check Codes. *Springer JSPS 2008*, DOI 10.1007/s11265-008-0231-5

[37] Daesun Oh, Keshab K. Parhi. Low Complexity Implementations of Sum-Product Algorithm for Decoding Low-Density Parity-Check Codes. *IEEE SIPS 2006*, pp. 265-267

[38] In-Cheol Park, Se-Hyeon Kang. Scheduling Algorithm for Partially Parallel Architecture of LDPC Decoder by Matrix Permutation. *IEEE ISCAS 2005*, pp. 5778-5781

[39] T. Richardson. *Modern Coding Theory*. Cambridge University Press, ISBN 0521852293, 2008

[40] Jin Sha, Zhongfeng Wang, Minglun Gao, Li Li. Multi-Gb/s LDPC Code Design and Implementation. *IEEE Trans. VLSI 2009,* vol. 17, pp. 262-268

[41] Xin-Yu Shih, Cheng-Zhou Zhan, Cheng-Hung Lin, An-Yeu (Andy) Wu. An 8.29 mm^2 52 mW Multi-Mode LDPC Decoder Design for Mobile WiMAX System in 0.13 μm CMOS Process. *IEEE JSSC 2008*, vol. 43, pp. 672-683

[42] M. Smith. *Application-Specific Integrated Circuits*. Addison Wesley. 1997. ISBN: 0201500221

[43] Yang Sun, Marjan Karkooti, Joseph R. Cavallaro. VLSI Decoder Architecture for High Throughput, Variable Block-size and Multi-rate LDPC Codes. *IEEE ISCAS 2007*, pp. 2104-2107

[44] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inf. Theory 1981*, vol. 27, pp. 533-547

[45] D. Thomas, P. Moorby. *The Verilog® Hardware Description Language*. Springer 2002. 5th ed., ISBN-13: 978-0387849300

[46] Jun Xu, Lei Chen, Lingqi Zeng, Lan Lan, and Shu Lin. Construction of low-density parity-check codes by superposition. *IEEE Trans. Communication 2005*, vol. 53, pp. 243-251

[47] E. Yeo, P. Pakzad, B. Nikolic, V. Anantharam. High Throughput Low-Density Parity-Check Decoder Architectures. *IEEE GLOBCOM 2001*, vol. 5, pp. 3019–3024

[48] Luoming Zhang, Lin Gui, Youyun Xu, Wenjun Zhang. Configurable Multi-Rate Decoder Architecture for QC-LDPC Codes Based Broadband Broadcasting System. *IEEE Trans. Broadcasting 2008*, vol. 54, pp. 226-235

[49] H. Zhong and T. Zhang. Design of VLSI implementation-oriented LDPC codes. *IEEE VTC 2003*, pp. 670-673

Index

D

E

M

N

O

P

Q

R

S

T

U

V

W