LDPC DECODER MODELLING AND EVALUATION USING RT-PEPA

Tony Tsang[1]

[1]Hong Kong Polytechnic University, Hung Hom, Hong Kong.
ttsang@ieee.org

ABSTRACT

This paper presents a high-throughput memory efficient decoder for Low Density Parity Check (LDPC) codes in the high-rate wireless personal area network application. The novel techniques which can apply to our selected LDPC code is proposed, including parallel blocked layered decoding architecture and simplification of the WiGig networks. We use Real Time - Performance Evaluation Process Algebra (RT-PEPA) to evaluate a typical LDPC Decoder system's performance. The approach is more convenient, flexible, and lower cost than the former simulation method which needs develop special hardware and software tools. Moreover, we can easily analysis how changes in performance depend on changes in a particular modes by supplying ranges for parameter values.

KEYWORDS

LPDC, IEEE 802.15.3c, RT-PEPA, Performance Analysis, Formal Modelling.

1. INTRODUCTION

Low density parity check (LDPC) codes, first proposed by Gallager in 1962 have attracted much attention because of their excellent error correcting performance, inherently parallelism and high throughput potentials. Therefore, they are being widely used in communication standards such as IEEE 802.16e and IEEE 802.11n. In addition, millimeter wave (mmWave) Wireless Personal Area Networks (WPANs) described by the IEEE 802.15.3c Working Group are considering LDPC codes as the preferred choice for forward error correction (FEC).

Recently, many studies have been accomplished to simplify the VLSI implementation of the related decoders called "Architecture-Aware LDPC codes" or "Block-LDPC codes". Based on these design approaches, quasi-cyclic LDPC (QCLDPC) codes have received significant attentions due to their efficient hardware implementations. Furthermore, QC-LDPC codes can provide comparable error-correction performance compared with random LDPC codes.

With the increasing demand for high-data-rate wireless application, an overlapped decoding architecture and a layered decoding architecture are popular for high-throughput LDPC decoding architectures using QC-LDPC codes. An overlapped decoding architecture based on the partially parallel architectures using the belief-propagation (BP) algorithm, where the message updating computations in the check node unit (CNU) and in the variable node unit (VNU) are partially overlapped, can increase the decoding throughput by maximizing the hardware utilization efficiency without any performance degradation.

Compared with the overlapped two-phase decoding scheme, a layered decoding algorithm and architecture have been proposed to achieve a faster convergence. Generally, layered decoding algorithm is the horizontal layered decoding algorithm which is favorable for the Min-Sum algorithm. The layered decoding algorithm offers 2x throughput and significant memory advantages, compared with the BP algorithm. Also, it can reduce the average number of iteration using intermediate check-node (or variable-node) message values. However, conventional layered decoders use a bidirectional network or two switch networks for shuffling and reshuffling messages, which increases the hardware complexity. Also, due to the data dependency between consecutive rows in layered decoding, the parallel and pipelining techniques cannot be applied directly. In this paper, we develop a fully pipelined architecture targeted for the IEEE WiGig standard that has fully parallelized variable nodes and layer serialized check nodes. By exploiting the structure of the lower rate codes, it reduces the number of sub-iterations by up to half.

The rest of the paper is structured as follows. Section II introduces Real-Time Performance Evaluation Process Algebra (RT-PEPA) and its Tools. Section III describes the design choices made for the architecture of Decoder Functional Blocks Specification, and Section IV details performance evaluation result of the functional design of the decoder's major blocks. Then, conclusion is presented in Section V.

2. RT-PEPA and its Tools

We present a formal modelling language, called Real-Time Performance Evaluation Process Algebra (RT-PEPA), to describe the real-time stochastic behaviour of communication systems. The language combines conventional stochastic process algebra with real-time semantics to describe complex systems in a compositional manner. It includes timed transition, parallel composition, probabilistic branching and hard real-time aspects. Performance Evaluation Process Algebra (PEPA), developed by Hillston in the 1994s [3, 4], is a timed and stochastic extension of classical process algebras such as Communication Sequential Process (CSP) [5]. It describes a system as an interaction of the components and these components engage in activities. Generally, components model the physical or logical elements of a system and activities characterize the behavior of these components. An exponentially distributed random variable is associated with each activity specifies the duration of it, that leads to a clear relationship between the model and a Continuous Time Markov Chain (CTMC) process. Via this underlying Continuous Time Markov Chain process performance measures can be extracted from the model. The PEPA formalism provides a small set of timed operators which are able to express the individual real time activities of components as well as the interactions between them. We provide a brief summary of the operators here, more details about PEPA can be found in [2, 3].

2.1. Notation and Definitions of RT-PEPA

Real-Time - Performance Evaluation Process Algebra (RT-PEPA) is process algebraic language which supports the compositional description of concurrent and distributed systems and analysis of their performances. The basic elements of RT-PEPA are its actions, which represent activities carried out by the systems being modeled, and its operators, which are used to compose algebraic descriptions.

Time point

A time point is a time instant with respect to the global clock of the real time system; it does not have duration. It specifies the starting and stopping times of an action. Using a time point, we can instruct the system to generate an action at a particular point in time. Time point progresses consistently in all parts of the system. More formally, the time point is defined by using a discrete time domain, which contains the following properties:

$\forall t \, \exists t' \, t < t' \wedge \forall t'' : t < t'' \Rightarrow t' \leq t''$

We assume a fixed set of clock $t = \{t_0, \ldots, t_i\}$. The special time point *t*0, which is called the start time point, always has the value 0.

Time Constraint

An action can exist for a short period of time; this duration is called the time constraint of the action. A time constraint has a starting and an ending point. It consists of a lower-bound and an upper-bound time point, where the lower-bound time point enables an action in a module, and the upper-bound time point disables the action at that point in time. Formally, we define a time constraint in the following:

$\mathrm{T}_i = \{[\tau_{i_{min}}, \tau_{i_{max}}] \ \forall t_i \in T\} \ with \ 0 \le \tau_{i_{min}} \le \tau_{i_{max}}$

Timed Action

A timed action is a tuple $<\alpha, \lambda, \mathrm{T}>$ consisting of the type of the action α, the rate of the action λ and temporal constraint of the action T. The type denotes the kind of action, such as transmission of data packets, while the rate indicates the speed at which the action occurs from the view of an external observer. The rates are used to denote the random variables specifying the duration of the actions. The actions can be defined in different types of probability distribution function such as Exponential, Poisson, Constant, Geometric and Uniform distribution. Moreover, each transition is also bounded by a temporal constraint. In this section, some basic notations and operation semantics about RT-PEPA are briefly introduced. The syntax of RT-PEPA is defined in the following:

$$P ::= stop \mid <\alpha, \lambda, \mathrm{T}>.P \mid P + Q \mid P \oplus_{r,\mathrm{T}} Q \mid P \ \otimes_{L,\mathrm{T}} Q \mid P \ \square_{\mathrm{P},\mathrm{T}} Q \mid P / L \mid A$$

The conventional stochastic process algebra operators and the additional operations are described in the following:

- *stop* is an inactive process
- $<\alpha, \lambda, \mathrm{T}>.P$, which stands for a prefix operator, where the type of the action is a probability distribution function (pdf) type α , with the activity rate denoted by λ , and the temporal constraint of component is T . It subsequently behaves as *P*. Sequences of actions can be combined to build up a time constraint for an action. The time constraint T is defined as above.
- $\mathrm{P} + \mathrm{Q}$ is choice combinator capturing the possibility of competition or selection between different possible activities. It represents a system which may behave either as *P* or as *Q* processes. All the current actions *P* and *Q* process are enabled. The first action to complete distinguishes one of the processes. The other process of the choice is discarded. The system will then behave as the derivative resulting from the evolution of the chosen process.
- $P \oplus_{r,\mathrm{T}} Q$ denotes the probabilistic choice with the conventional generative interpretation, thus with probability *r* the process behaves like *P* and with probability $1 - r$ it behaves like *Q* bounded with the time constraint T .
- $P \ \otimes_{L,T} Q$ is a cooperation, in which the two actions *P* and *Q* are parallel, synchronizing on all activities whose type is in the cooperation set *L* of action types. The lifetime of two actions is the time constraint T . These two actions are disabled when the time constraint expires. Any action whose type is not in *L* will proceed independently. As a syntactic convenience the parallel combinator is defined by $\otimes_{\phi,\mathrm{T}}$, where the cooperation set *L* is empty and the lifetime of two actions is T .

- $P \; \square_{\mathrm{P},\mathrm{T}} \, Q$ is a unary operator which returns the set of actions that meet the temporal predicate condition specified by T . P consists of several predicates combined with the boolean connectives: `And' ,`Or', Exclusive-Or (EXOR)' and `Not'. $\square_{\mathit{And},\mathrm{T}}$ means both actions can occur during the interval T . $\square_{\mathit{Or},\mathrm{T}}$ means that one or both actions can occur during the interval T . $\square_{\mathit{EXOR},\mathrm{T}}$ means that one of these actions occurs; it immediately determines whether *P* or *Q* can subsequently occur during the triggered interval T . $\square_{\mathit{Not},\mathrm{T}}$ means that both actions do not occur during the interval T .

- $P \,/\, L$ is a hiding operation, where the set *L* of visible action types identifies those activities which are to be considered internal or private to the component. These activities are not visible to an external observer, nor are they accessible to other components for cooperation.

- $A := P$ is a countable set of constants.

2.2. RT-PEPA Eclipse Plug-in Tools

The PEPA is a language for modelling systems in which a number of interacting components run in parallel, and whose behaviour is stochastic. The core semantics of PEPA is in terms of Continuous Time Markov Chains (CTMCs), and an alternative semantics in terms of Ordinary Differential Equations (ODEs) has also been developed. PEPA has been applied in practice to a wide variety of systems, and its success as a modelling language has been largely down to its extensive tool support. Most recently, the PEPA Plug-in Project [6, 7] has integrated a range of analysis techniques based on both numerical solution and simulation into a single tool built on top of the Eclipse platform [8]. As with all compositional Markovian formalisms, however, PEPA suffers from the state space explosion problem. A model can have an underlying state space that is exponentially larger than its description, meaning that it can be infeasible to analyse. Fluid flow approximation using PEPAs ODE semantics can solve this problem if we are only interested in the average behaviour of the system over time. However, if we want to reason over all possible behaviours of the model for example, the probability that an error occurs within some time interval then we must consider the CTMC semantics. In this paper, we present a new extension to the PEPA plugin, in which a model can be abstracted by combining, or aggregating, states. To safely over-approximate the behaviour of the original model (for any aggregation of its states), we use two abstraction techniques - abstract CTMCs (a type of Markov decision process with infinite branching), and stochastic bounds. We provide a model checker for the three-valued Continuous Stochastic Logic (CSL), which computes from the abstraction a safe bound of the probability of a quantitative property holding in the original model X if the actual probability is p, then the model checker will return an interval I = [p1, p2] such that p 2 I. The current version of the PEPA plug-in is available from http://www.dcs.ed.ac.uk/pepa/tools/plugin, and provides several views:

2.2.1 Abstract Syntax Tree View

The Abstraction View is a graphical interface that shows the state space of each sequential component in a PEPA model. It provides a facility for labelling states (so that they can be referred to in CSL properties), and for specifying which states to aggregate.

2.2.2 Model Checking View

The Model Checking View is an interface for constructing, editing, and model checking CSL properties. The property editor provides a simple way to construct CSL formulae, by referencing the labels given to states in the abstraction view. It ensures that only syntactically well-formed CSL formulae can be constructed.

2.2.3 State Space View

The State Space View is linked to the active PEPA editor and provides a tabular representation of the state space of the underlying Markov chain. The table is populated automatically when the state space exploration is invoked from the corresponding top level menu item. A row represents a state of the Markov chain, each cell in the table showing the local state of a sequential component. The state space order in which sequential components are displayed corresponds to the order in which they are found in the co-operation set by depth-first visit of the co-operations binary tree. A further column displays the steady-state probability distribution if one is available. A toolbar menu item provides access to the user interface for managing state space filters. When a set of filter rules is activated, the excluded states are removed from the table. The probability mass of the states that match the filters is automatically computed and shown in the view. Filter rules are assigned names and made persistent across workspace sessions. From the toolbar the user can invoke a wizard dialogue box to export the transition system and one to import the steady-state probability distribution as computed by external tools. The view also has a Single-step Debugger, a tool for navigating the transition system of the Markov chain. The debugger can be opened from any state of the chain and its layout is as follows. In an external window are displayed the state description of the current state and two tables. The tables show the set of states for which there is a transition to or from the current state. The tables are laid out similarly to the views main table. In addition, the action types that label a transition are shown in a further column. The user can navigate backwards and forwards by selecting any of the states listed.

2.2.4 Performance Evaluation View and Graph View

Performance Evaluation View and Graph View A wizard dialogue box accessible from the top-level menu bar guides the user through the process of performing steady-state analysis on the Markov chain. The user can choose between an array of iterative solvers and tune their parameters as needed. Performance metrics are calculated automatically and displayed in the Performance Evaluation View. It has three tabs showing the results of the aforementioned reward structures (throughput, utilisation, and population levels). Throughput and population levels are arranged in a tabular fashion, whereas utilisation is shown in a two-level tree. Each top-level node corresponds to a sequential component and its children are its local states. The Performance Evaluation View can feed input to the Graph View, a general purpose view available in the plug-in for visualising charts. Throughputs and population levels are shown as bar charts and a top-level node of the utilisation tree is shown as a pie chart. As with any kind of graph displayed in the view, a number of converting options is available. The graph can be exported to PDF or SVG and the underlying data can be extracted into a comma separated value text file.

2.2.5 Experimenting with Markovian Analysis

An important stage in performance modelling is sensitivity analysis, i.e. the study of the impact that certain parameters have on the performance of the system. A wizard dialogue box is available in the plug-in to assist the user with the set-up of sensitivity analysis experiments over the models. The parameters that can be subjected to this analysis are the rate definitions and number of replications of the array of processes in the system equation. The performance metrics that can be analysed are throughput, utilisation, or population levels. If the model has filter rules defined, the probability mass of the set of filtered states can be used as a performance index as well. The tool allows the set-up of multiple experiments of two kinds: one-dimensional (performance metric vs. one parameter) or two-dimensional (performance metric vs. two parameters changed simultaneously). The results of the analysis are shown in the Graph View as line charts. For example, a parameter that may have an important impact on the performance of the real time system is the reset delay of the CPU.

2.2.6 Time Series Analysis

When performing a time-series analysis there are three basic steps to complete; component selection, solver selection and solver parameterization, all of which are handled by the time-series analysis wizard. Rather than simply observing all components, the wizard allows the modeller to select only those components that are of interest. This becomes more pertinent as either the number components in the system or number of observed time points increase - one limitation of the current time-series solvers is that all data is held in memory, and only written out to disk when exporting from the graph view. Solver selection and parameterization are self-explanatory, with the list of visible parameters being dynamically linked to the currently selected solver. In keeping with the rest of the UI, the selections across all three steps are persistent across invocations. Likewise, each unique parameter is stored only once, meaning parameters such as start and stop times are persistent over all solvers. Lastly, the parameters, including selected solver, are attached to the results in the graph view for future reference. Currently this meta data can only be seen when the data is exported. The last feature of the wizard is the ability to export the model in alternative formats, such as Matlab.

3. Decoder Functional Blocks Specification

3.1. Overall System

The decoder has five major blocks: variable nodes (VN), check nodes (CN), barrel shifters (BS), pre-routers, and post-routers. Figure 1 shows the high-level connection of all the blocks. The VNs are combined into 16 groups of 42VNs, called a variable node group (VNG). Each VN within a VNG connects to one port of a 42-input barrel shifter to implement the sub-matrix shifts. The outputs of the barrel shifter are further routed by the pre-routers, which connect to one of the 16 inputs of the 42 CNs. The outputs of each CN go through inverse shifting using post-routers and another set of barrel shifters. The design has several levels of hierarchy in order to keep irregular wiring local and make the global wires as regular as possible. Since the design layer serializes the CNs, they are time multiplexed to act as different CNs in each cycle. The CN has all of the information it needs to compute the new check to variable (C2V) message from all CNs in the first layer, and, after it has finished processing the inputs, it sends back a single message to all neighboring VNs. In the next cycle, the VNs send another single message that has been marginalized for the CNs in the second layer, so that the same CNs can compute the C2V message from the CNs of the second layer. This continues for all the layers in the matrix, with serial messages being passed back and forth from VNs to CNs. The decoder uses the flooding schedule because layered decoding has too many dependencies to be used effectively in a highly parallel, fully pipelined design. It processes one layer per pipeline stage, and the VNs accumulate one frames messages while sending out the others. The decoder equation defines how the components interact with each other. According to the working cycle and the definitions of model's components we give before, the decoder equation is show below:

$$\begin{aligned}&Decoder_0 := Back-Shifter_0 \otimes_{L,T} Variable-Node_0 \otimes_{L,T} Front-Shifter_0 \otimes_{L,T} \\ &\quad Pre-Router_0 \otimes_{L,T} Check-Node_0 \otimes_{L,T} Decoder_0 ;\end{aligned}$$

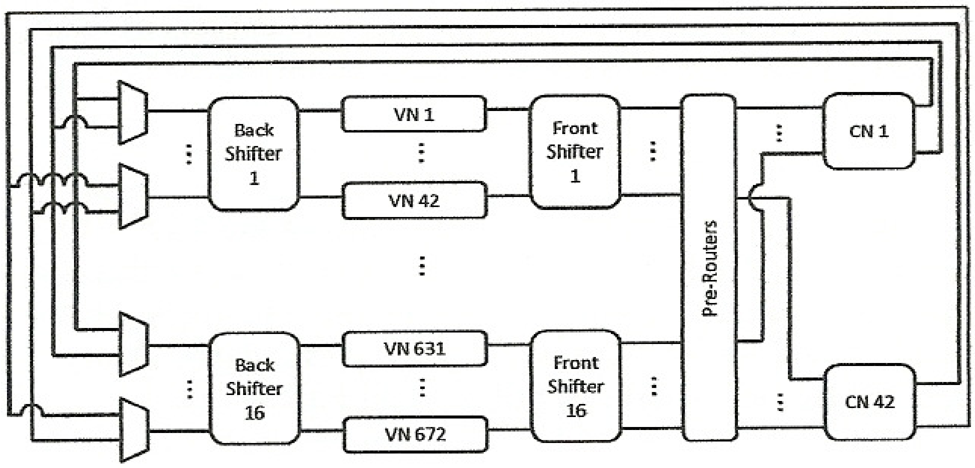

Fig. 1. Decoder Block Diagram

3.2. Variable Node

The VN implements [9] and performs both the CN and VN marginalization, which keeps all memory within the VN. When starting to decode a new data frame, the VN loads in a prior value to either the first or second frames prior or accumulation registers. Over the next four cycles sends variable to check (V2C) messages to its neighboring CNs. Since the VN performs the C2V marginalization, it locally stores the V2C messages it just sent out in a shift register. Once the un-marginalized C2V message returns, the VN compares the first minimum magnitude to the V2C magnitude from the shift register. If they match, the VN uses the second minimum; otherwise, it uses the first minimum. It marginalizes the sign by taking the XOR of the new and stored V2C signs. The VN accumulates the marginalized C2Vs, with the prior value added in when the first C2V arrives, and it also stores them individually in another shift register. After all C2V messages have been accumulated, the VN calculates the new V2C messages by subtracting out the saved C2V values from the accumulated value, which is the V2C marginalization. Also, the sign bit of the accumulated value is used as the hard decision and for early termination. Figure 2 shows the block diagram for the VN.

$$\begin{aligned}&Variable-Node_0 := \beta Corrector_0 \otimes_{L,T} S/M-2'sComp_0 \otimes_{L,T} XOR_0 \otimes_{L,T} \\ &\quad Accum_0 \otimes_{L,T} Subtractor_0 \otimes_{L,T} 2'sComp-S/M_0 ;\end{aligned}$$

$$\beta Corrector_0 := < listen, \lambda_0, \mathrm{T}_0 > .C2V-Min_0 \otimes_{L_0} < get, \lambda_0, \mathrm{T}_0 > .Mux_0 ;$$

$$Subtractor_0 := < listen, \lambda_0, \mathrm{T}_0 > .Mux_0 \otimes_{L_0} < get, \lambda_0, \mathrm{T}_0 > .V2C-Marg_0 ;$$

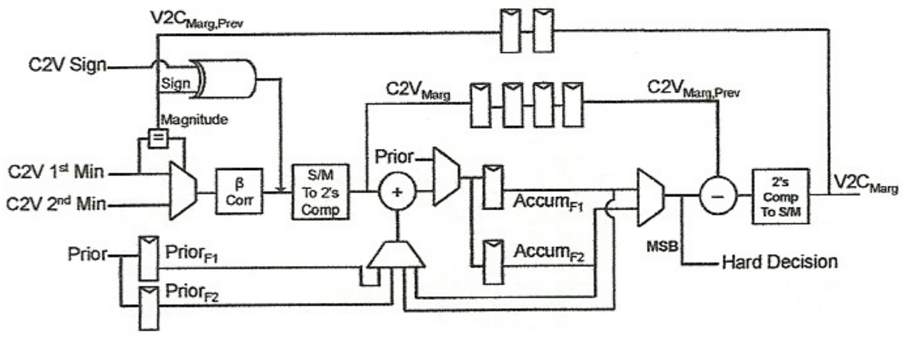

Fig. 2. Variable Node Schematic

3.3. Check Node Node

Since the VN performs the marginalization, the CN only needs to find the first and second minima of all the incoming V2C message magnitudes and the product of the V2Cs signs. The CN computes the former with a compare select tree. A sorting block at the beginning arranges pairs of inputs in ascending order, and subsequent stages of the tree select the minimum two out of four inputs. A separate XOR tree finds the product of the signs. The CN is modified to process either a single high-weight layer or two non-overlapping layers. To process a single layer, it takes the output of the entire tree to find the first and second minima of all 16 inputs. For non-overlapping layers, the weight is at most 8, so the top half of a tree can process one layer and the bottom half can process the other. The outputs are taken at the second to last stage to obtain first and second minima for each layer. This partitions a 16-input CN into two 8-input CNs, giving the CN two levels of granularity. Because the layers do not overlap, no read before write conflicts occur. Figure 3 shows the final CN architecture for the magnitude computation. The sign's XOR tree is partitioned similarly.

$$Check-Node_0 := Sort_0 \otimes_{L,T} CS_0 \otimes_{L,T} CS_1 \otimes_{L,T} XOR_0 \otimes_{L,T} CS_2 \;;$$

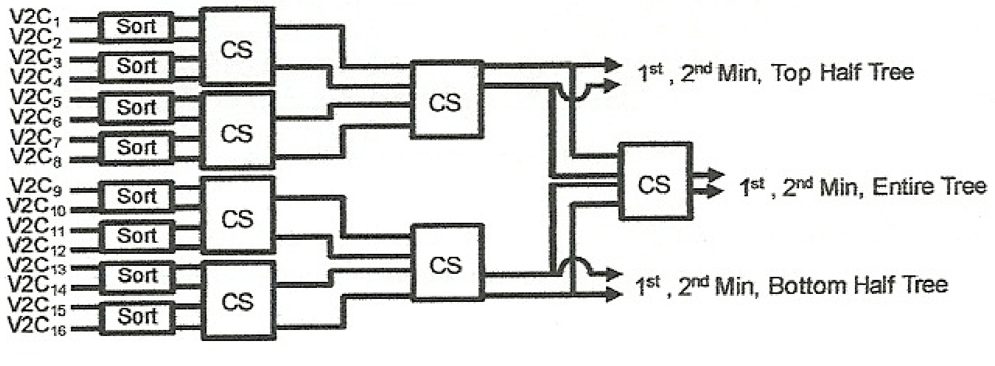
V2C1
V2C2
V2C3
V2C4
V2C5
V2C6
V2C7
V2C8
V2C9
V2C10
V2C11
V2C12
V2C13
V2C14
V2C15
V2C16
Sort
CS
1st, 2nd Min, Top Half Tree
1st, 2nd Min, Entire Tree
1st, 2nd Min, Bottom Half Tree

Fig. 3. Check Node Magnitude Computation Schematic

3.4. Pre- and Post-Routing

When processing two non-overlapping layers at once, barrel shifters alone cannot guarantee that the first layers inputs will go to the top half of each CN and the second layers inputs will go to the bottom half. Granular CNs need two extra sets of routing to allow this, as depicted in Figure 4. Pre-routing comes before the CN and selects which VNGs go to the top half or bottom half of each CN. Post-routing comes after the CN and, for each VNG, selects whether to send the top trees or bottom trees result. Both types of routers are implemented with a small number of muxes. In the case of processing one layer, the routers do not shift the messages.

$$Router_0 := Select_0 \otimes_{L,T} Pre-Router_0 \otimes_{L,T} Check-Node_1 \otimes_{L,T} Post-Router_0 \otimes_{L,T} Select_1 ;$$

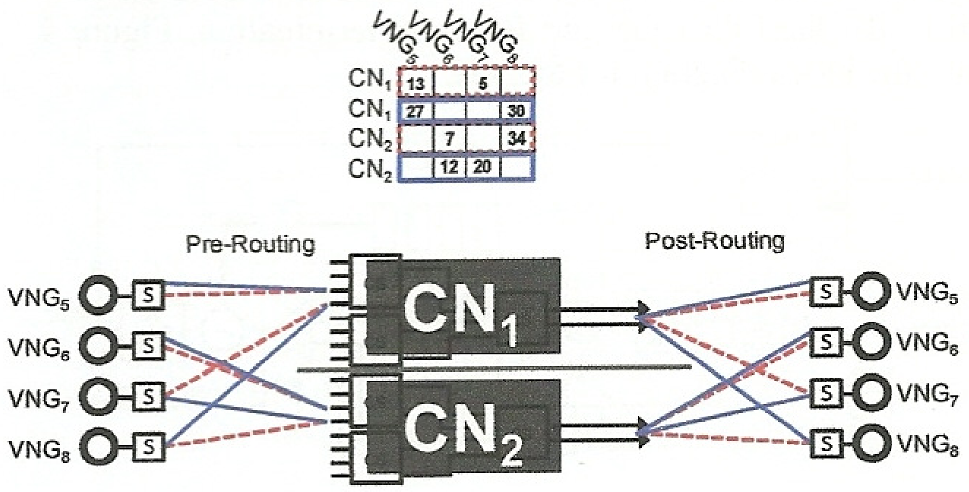

Fig. 4. Operation of Pre- and Post-Routers

3.5. Pipeline

The decoder has five pipeline stages and processes two independent data frames simultaneously. Each full iteration takes three sub-iterations for the rate 13/16 code and four for the rest, one for each time-multiplexed use of the check nodes. In the first stage, the VN outputs the marginalized V2C and the barrel shifter reorders the messages. The second stage consists of the pre-routing and global wiring to the check node. The CNs processes their inputs in the third stage and route the messages back to the VNs across the global wires in the fourth stage. In the fifth stage, the VN accumulates the serial messages over three or four cycles. The accumulation would normally cause a four cycle bubble in the pipeline, it due to the dependency between accumulating all the C2Vs and sending out the next V2Cs. The bubble is just large enough to accommodate processing a second frame. This reduces the bubble to one cycle for the rate 13/16 code and removes it completely for the other rates because no dependencies exist between the two frames. Figure 5 gives the pipeline diagram for the decoder with four sub-iterations.

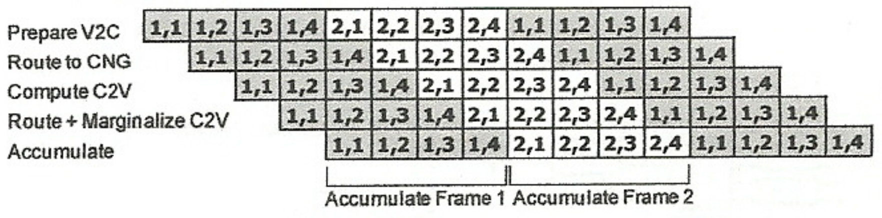

Fig. 5. Pipeline Diagram for rate 1/2, 5/8, and 3/4 codes where the first number indicates the frame and the second the sub-iteration

4. Performance Evaluation

The study of how changes in performance depend on changes in parameter mode values is known as sensitivity analysis. We can vary some parameter's value a little, and see its influence degree to the model's performance, for example, the throughput or response time. Throughput is an action-related metric showing the rate at which an action is performed at steady-state. In other words, the throughput represents the average number of the activities completed by the system during one unit time. From Figure 6, it can be observed that the impact of the number of devices on the throughput of transmit is more sensitive than the QPSK-1/2 and 16-QAM-1/2 modulation for NLOS channel CM 2.3 and LOS channel CM 1.2 Modes. If we could make some efforts to optimize the cache, and raise the 16-QAM-1/2 modulation for NLOS channel Mode form 0.6 to 0.85 or even more high value, the throughput of transmit could greatly improved.

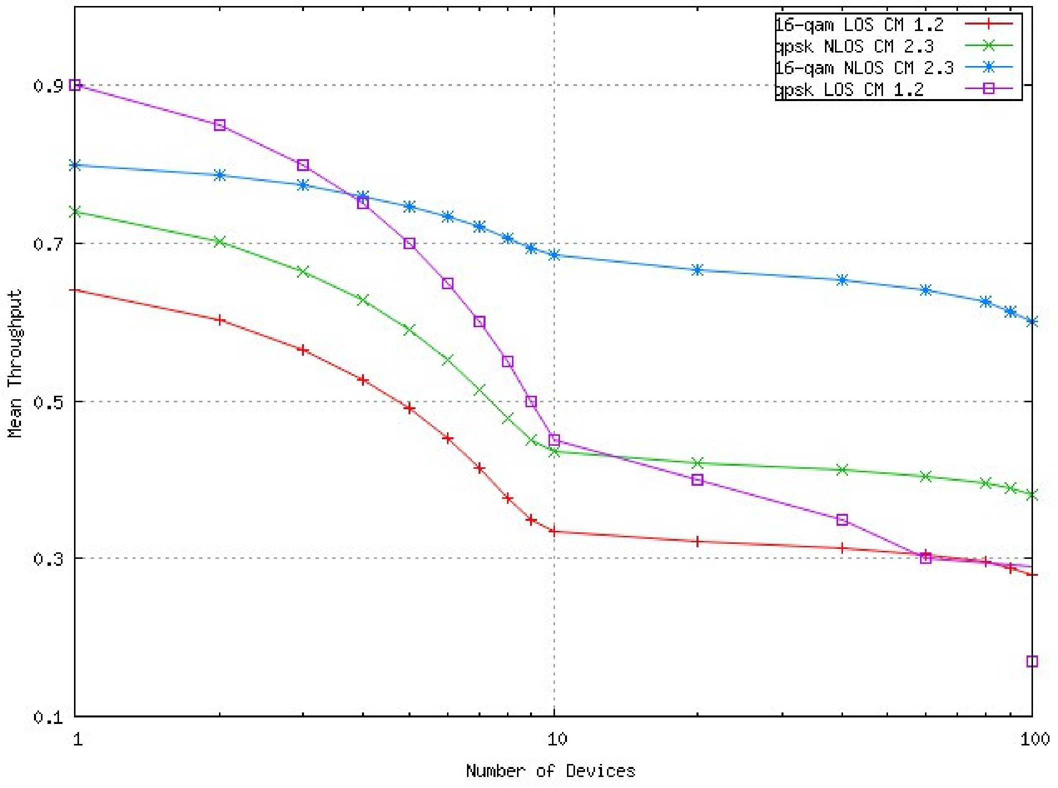

Fig. 6. Throughput versus Number of Devices

For channel coding, we have investigated the performance of LDPC (768,384) coded OFDM systems All receiver functions including packet detection, fine timing synchronization, channel estimation equalization and common phase-error correction have been included. Quantization of bit metrics has been performed with 5 soft-bits in both coding schemes. The chosen LDPC decoding algorithm is the min-sum algorithm with a maximum of 25 iterations per LDPC block. The performance for QPSK-1/2 and 16-QAM-1/2 modulation for NLOS channel CM 2.3 and LOS channel CM 1.2 are presented in Figure 7. The simulation results show better performance of LDPC code for both channel models. The target for the Bit Error Rate (BER) ranges from 10% down to 1%. In the presence of NLOS channel model (CM) 2.3, LDPC coding achieves higher coding gain, in the range of 1.6 to 2.7 dB for QPSK, and 1.0 to 2.5 dB for 16-QAM. On the other hand, in the presence of LOS channel CM 1.2, the coding gain of the LDPC code is approximately 1 dB.

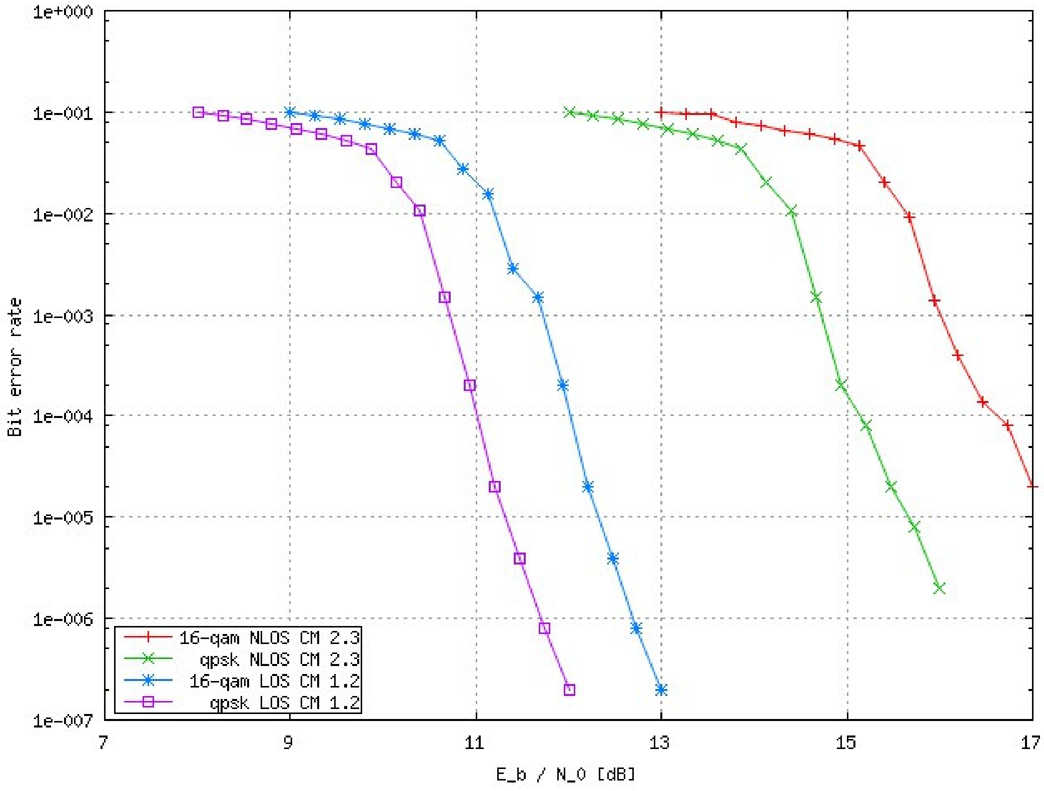

Fig. 7. BER of designed LDPC Decoder for 60-GHz NLOS Channel CM2.3 & LOS channel CM 1.2

5. CONCLUSIONS

The proposed fully pipelined architecture with granular check nodes supports all matrices and low-power modes of the IEEE 802.11ad standard. The routing complexity was reduced significantly by replacing a crossbar based interconnect network with a fixed wire network for SN. Hence, the proposed decoder architecture has high throughput, low interconnect complexity and very low decoding latency. The architecture can be extended to other short block length codes that have the same property of non-overlapping layers, such as to be incorporated in next-generation high-rate WPAN applications.

REFERENCES

[1]. IEEE Std 802.15.3c-2009 (Amendment to IEEE Std 802.15.3-2003), "IEEE Standard for Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks - Specific Requirements. Part 15.3: The ultimate purpose of the 60-GHz WPAN systems is to deliver MAC throughput of the order of multi-Gb/s over a reasonable range. To accomplish this, system designers have to increase the transmission range, especially in non-line-of-sight channels. IEEE Communications Magazine E July 2011 121 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", pps. c1-187, Octaber. 2009.

[2]. Tony Tsang, "Performance modelling and evaluation of OFDMA based WiMAX systems using RT-SPA", Proceedings of the International Conference on Computer and Communication Engineering 2008 (ICCCE08), Kuala Lumpur, Malaysia, pps. 180- 186, May 13-18, 2008.

[3]. J. Hillston, "A Compositional Approach to Performance Modelling", PhD Thesis, The University of Edinburgh, 1994.

[4]. J. Hillston, "Fluid flow approximation of PEP A models", Proceedings of the Second International Conference on the Quatitative Evaluation of Systems, IEEE Computer Society Press, pps. 33-41, 2005.

[5]. C.A.R.Hoare, "Communicating Sequential Process", Prentice-Hall, 1985.

[6]. Micheal J.A. Smith, "Abstraction and Model Checking in the PEPA plug-in for Eclipse", Seventh International Conference on the Quantitative Evaluation of Systems, pps. 155-156, 2010.

[7]. M. Tribastone, A. Duguid, and S. Gilmore. "The PEPA Eclipse Plug-in", Performance Evaluation Review, 36(4):28-33, March 2009.

[8]. The Eclipse platform. http://www.eclipse.org.

[9]. Weiner Matthew; Nikolic Borivoje; Zhang Zhengya; "LDPC decoder architecture for high-data rate personal-area networks", IEEE International Symposium on Circuits and Systems (ISCAS), pps. 1784 - 1787, 15-18 May, 2011.

Tony Tsang

received the BEng degree in Electronics & Electrical Engineering with First Class Honours in U.K., in 1992. He received the Ph.D from the La Trobe University (Australia) in 2000. He was awarded the La Trobe University Post-graduation Scholarship in 1998. He is a Lecturer at the Hong Kong Polytechnic University. Prior to joining the Hong Kong Polytechnic University, Dr. Tsang earned several years of teaching and researching experience in the Department of Computer Science and Computer Engineering, La Trobe University. His research interests include mobile computing, networking, protocol engineering and formal methods. Dr. Tsang is a member of the ACM and the IEEE.