Parallel decoder for Low Density Parity Check Codes: A MPSoC study

Sudeep Kanur Chandra Shekar
Åbo Akademi
Turku, Finland
Email: skanur@abo.fi

Georgios Georgakarakos
University of Turku
Turku, Finland
Email: geogeo@utu.fi

Antti Siirilä
University of Turku
Turku, Finland
Email: anjosi@utu.fi

Jérémie Lagravière
University of Turku
Turku, Finland
Email: jerlag@utu.fi

***Abstract*—The near channel performance of Low Density Parity Check Codes (LDPC) have motivated its wide applications. Iterative decoding of LDPC codes provides significant implementation challenges which grows with the code size. Recent trends in integrating Multiprocessor System on Chip with Network on Chip gives a modular platform for parallel implementation of decoders. This paper uses HeMPS, an open source MPSoC framework based on NoC communication fabric to implement reduced minimum sum algorithm and measures the throughput and speed-up factor.**

***Index Terms*—Low Density Parity Check codes, HeMPS, MPSoC, NoC, Message passing interface (MPI)**

I. Introduction

Reliable transmission of data over a noisy communication channel is one of the central goal a communication system strives to achieve. Forward Error Correction schemes are widely used in broadcasting and communication systems to increase the bandwidth and improve the efficiency. Low Density Parity Check (LDPC) codes comes under the class of forward error correction codes and was introduced by Gallager [1]. LDPC codes have been proved to achieve performance close to Shannon's limit [2] and this performance has motivated its use in application areas ranging from long range satellite transmission to terrestrial broadcasting [3].

LDPC codes belongs to linear block codes and enjoys its properties such as improved performance with large blocks of information and relatively simple encoding procedure. However, decoding LDPC codes is a NP-complete problem [4]. Iterative decoding schemes are used in decoding, but they provide significant implementation challenges. However on the positive side iterative decoding schemes provide modularity for parallel implementation which many implementation platforms have exploited for faster throughput. The work in [5] and [6] presents IP cores for DVB-S2 standards on an ASIC, while [7] present the same on flexible platforms such as field programmable gate arrays (FPGAs). Software Defined Radio (SDR) and GPU implementation have also been implemented for the standard [8]. While afore mentioned work have achieved real time throughput rates, they come at the cost of increased design time as the IP cores are hand-coded in the cases of FPGA or increased size of the hardware as in the case of GPUs and SDR.

With the advancements in semiconductors and CAD design, Multiprocessor System on Chip (MPSoC) with underlying Network on Chip (NoC) platform is gaining popularity in embedded systems. The platform provides modularity at processor level and can be exploited for the implementation of iterative decoding of LDPC. HeMPS is an open source framework targeting MPSoC customisation that includes platform comprising of NoC, processors and DMA, embedded software comprising of microkernel and applications and a dedicated CAD tool to generate required binaries for implementation on FPGA platform and perform debugging [9]. Communication between processors are done using NoC communication structure and is implemented using message passing routines. In addition the platform supports static and dynamic task mapping and C/SystemC simulation models for processors and memories [10].

Exploitation of platform's modularity coupled with decreased time and cost of implementation is the main goal of the paper. The decoding of LDPC code is performed on a message length of 504 bits described by a rate 1/2 parity matrix of size 252 rows and 504 columns with progressive edge growth construction. The work simulates sequential and parallel implementation of the algorithm and measures throughput data rates. A scale up factor is also presented and data rates are compared with an MPI implementation on a desktop machine to get a perspective. The paper is organised as follows. Section II provides details on the decoding algorithm. Section III introduces the MPSoC framework. Section IV presents the experimental setup and details mapping of algorithm on the framework. Section V presents observations and results and concludes the paper.

II. Decoding of LDPC codes

A LDPC code of length n bits consists of k bits of information and $n-k$ bits of redundancy called parity bits. The code rate $R = k/n$ gives fraction of useful information sent by the code. The relationship between information and parity bits are given by linear equations and these linear equations can be represented in a matrix form, known as parity check matrix H. A parity check matrix for a Hamming code with code length of 7 and information bits of 4 is shown in fig.1. Bipartite graphs can be constructed from parity check matrix H by connecting *check nodes*, given by row indices of H with *variable nodes*, given by column indices of H and is drawn according to the following rule: a check node i can connect to

variable node j whenever the element h_{ij} of H is 1, where i & j represent rows and columns of parity matrix. This graphs shown in fig.2, known as *Tanner graphs*, represent the entire LDPC code and can aid in understanding decoder algorithms for LDPC codes.

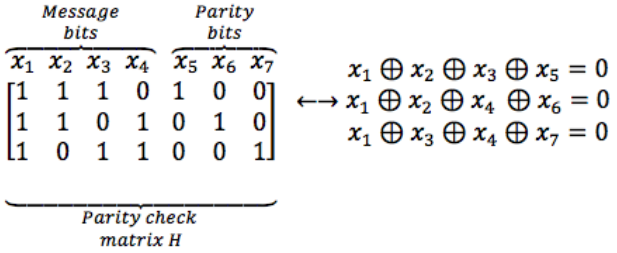

Fig. 1. Parity Check Matrix, H

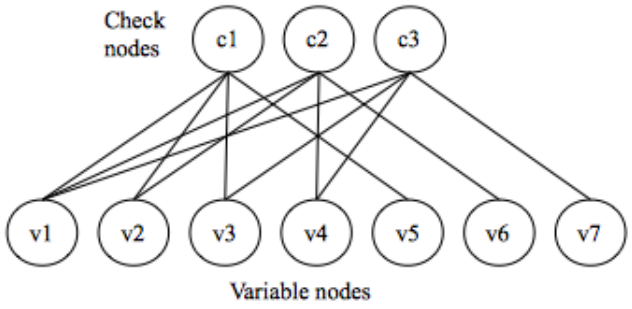

Fig. 2. Tanner graph representation of parity matrix H

Gallager proposed an iterative probabilistic decoding scheme based on message or belief propagation to decode LDPC codes [1]. Several decoding techniques have been proposed since then and all of effective decoding strategies for LDPC are message passing algorithms. Minimum Sum algorithm has been proved to provide good performance with usage of minimal hardware [11]. Reduced scan minimum sum algorithm, a variation of minimum sum algorithm, was proposed for systems with memory constraints [12]. This algorithm is a rearrangement of the equations of minimum sum algorithm and has less memory footprint than the original algorithm. The reduced footprint come at the cost of reduced performance, and in implementation constrained with memory, this algorithm can be used.

Reduced minimum sum algorithm (RMSA) works by passing messages or beliefs between the check nodes and variable nodes of the tanner graph. RMSA operates in logarithm domain and messages passed between nodes known as Log Likelihood Ratios (LLRs) are given by preceding receiver stage. LLR is given by,

$$\Lambda = log(\frac{P(r|s=0)}{P(r|s=1)}), \qquad -\infty \leq \Lambda \leq \infty \tag{1}$$

LLRs exhibit the property that as $\Lambda \to \infty$, $\hat{s} \to 0$ and as $\Lambda \to -\infty$, $\hat{s} \to 1$, where $\hat{s}$ is the decoded message.

Let us consider a tanner graph shown in fig.2 where $C(v)$ denotes the set of check nodes which connect to variable nodes v. Similarly let $V(c)$ denote the set of variable check nodes that connect to check nodes c. Let $C(v) \setminus c$ represent all members of set $C(v)$ except c, while $V(c) \setminus v$ represent all members of set $V(c)$ expect v. The RMSA decoding algorithm for i iterations can be summarised in following steps.

1) *Initialisation*: Each codeword symbol r shown by variable nodes of tanner graph are initialised by *priori* LLR. For each variable node v, assign

$$\Lambda_v^{(0)} = \Lambda^{(0)} \quad and \quad \Lambda_{c\to v}^{(0)} = 0$$

2) *Check node update*: For each check node c and for each $v \in V(c)$, compute

$$\Lambda_{c\to v}^{(i)} = (\prod_{v' \in V(c)\setminus v} sign(\Lambda_v^{(i-1)} - \Lambda_{c\to v'}^{(i-1)})) \times \min_{v' \in V(c)\setminus v} |\Lambda_v^{(i-1)} - \Lambda_{c\to v'}^{(i-1)}| \tag{2}$$

3) *Variable node update*: For each variable node v, compute

$$\Lambda_v^{(i)} = \Lambda^{(0)} + \sum_{c \in C(v)} \Lambda_{c\to v}^{(i)} \tag{3}$$

4) *Decision*: Obtain intermediate $\hat{r} = 0$ if $\Lambda_v^{(i)} \geq 0$ and $\hat{r} = 1$ otherwise. Check for the condition $H \cdot \hat{r}^T = 0$. If the condition is satisfied $\hat{r}$ is a valid codeword and $\hat{s}$ can be extracted, else goto step 2 and iterate until iteration limit is reached.

Careful observation of the algorithm at step 2 reveals that the check node computation of a given check node is independent of others and all the check node computations can run in parallel. Hence, in an ideal case, all the check nodes can be made to run in parallel with messages passing between check nodes and variable nodes.

III. HeMPS framework

HeMPS frame work provides an MPSoC framework with network on chip like connectivity. The HERMES NoC architecture is used to connect "Plasma" (MIPS-like) processors in a 2D mesh topology as shown in fig. 3. The processing elements (PEs) are named Plasma-IP. Each PE in the network contains a plasma processor, a local memory, a DMA controller, and a network interface. The DMA controller is used for fast data transfer of packets between local memory and the PE's network interface. Among the PEs one 'master' (PlasmaIP-MS) is responsible for managing resources while the rest 'slaves' (PlasmaIP-SL) are executing the applications.

During operation, the master reads the distributed application source (split in several tasks) from an external task repository, and allocates tasks to the slaves. Each slave runs a microkernel, which supports multitasking and task communication. The master also runs a microkernel, but does not execute applications tasks. The microkernel segments the memory in pages, which is allocated for itself (first page) and several tasks (subsequent pages). Each Plasma-IP has

a task table, with the location of local and remote tasks. The microkernel protects memory pages. All communication among tasks is handled through a custom blocking message passing interface. The messages exchanged between cores are constrained to 128 bytes each. The kernel is described mostly in C and some special functions such as interruption treatment and context saving are described in assembly.

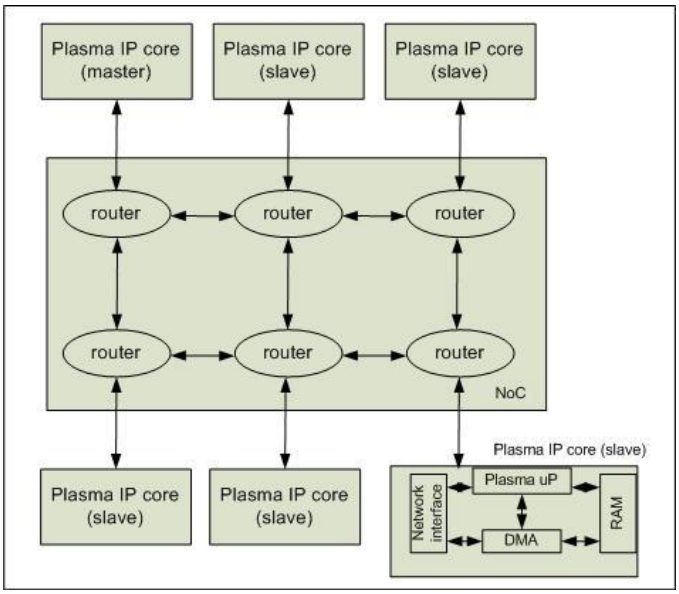

Fig. 3. HeMPS NOC architecture

IV. Implementation

A. Mapping

For a parallel scalable implementation of decoding algorithm, the option of assigning each node to a separate processing element is clearly not attractive, due to the high communication load between the processing elements when compared to the computational load of each node. Instead, the approach proposed is to group several check nodes together and to execute each group on separate processing elements in a homogenous MPSOC architecture.The goal is to have scalability and stable performance enhancements in LDPC decoding. In principle, the mapping involves dividing the decoding algorithm between one master processor and several slave processors. The master processor is in charge of scattering and gathering of data and computation of step 1, 3 and 4 of the algorithm while slave processors are in charge of unpacking the data and processing the check nodes.

The check node grouping decision is made by dividing the number of check nodes by the number of slave processors. Since the processing follows a homogeneous concept, each slave processor is assigned the same number of check nodes. Consider the tanner graph shown in fig.4. For the sake of convenience, 4 by 7 parity check matrix is chosen. Fig. 5 is an example of an implementation of the algorithm executed on 3 processing elements (*CPU0, CPU1 and CPU2*). *CPU0* executes all the variable nodes, and communicates with all

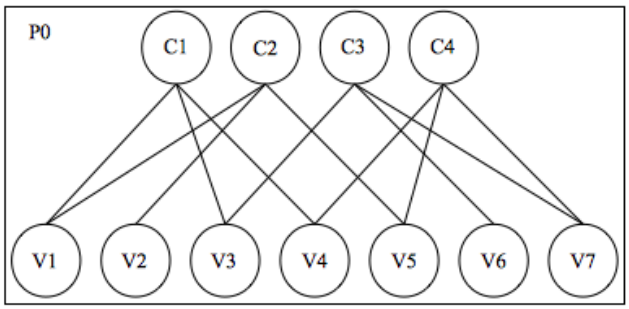

Fig. 4. Parallel programming model for LDPC decoder

the check nodes; in this example *CPU0* executes the central process. *CPU1* and *CPU2* are in charge of the execution of 2 check nodes each. Fig.6 shows the same for a generalised case.

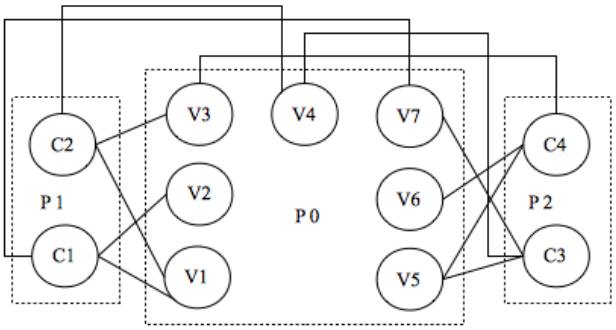

Fig. 5. Parallel programming model for LDPC decoder

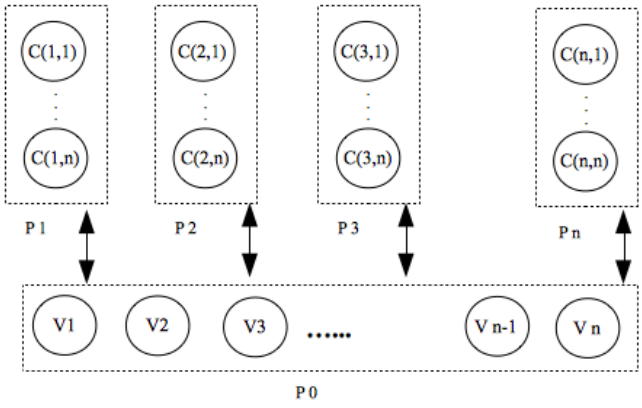

Fig. 6. Parallel programming model for LDPC decoder

B. Setup

The distributed LDPC decoder application is mapped as described in the previous section and transferred to HeMPS simulation platform for test and evaluation. For the experimentation, a parity check matrix of size 252 by 504 was chosen

[13]. A larger code length was favoured, however the simulator platform restricts the memory, and due to this restriction this parity matrix was chosen. To get a perspective of HeMPS platform, a measurement was carried out on a desktop machine using MPI as framework. This gives an idea of overheads that HeMPS platform puts on blocking message passing interface.The desktop machine has the following property: two CPUs *Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz*, in total it represents 12 physical cores, and 24 cores (hyperthreading), running *Linux Ubuntu 12.04 - Kernel 3.2.0-38-generic-pae* operating system.

HeMPS API allows various mapping options in a configurable NoC based MPSoC environment shown in fig. **??**. Since the mapping approach proposed features a master kernel communicating with several slave kernels, the master kernel is place at the center of the processing elements array, while the slave kernels are allocated in processing elements (PE) around the central PE. The PEs are configured as Plasma IP cores at a clock speed of 100 MHz and a maximum memory page size of 16KB (program and data). Each LDPC decoder kernel uses 1 memory page size per core. The input of the decoder is the LLR values of a single word from the channel receiver. The LLRs are hard coded into the memory page of the master node. Using a system call that measures clock cycles, a performance evaluation of several check node grouping options can be obtained. The measurements performed here are worst case measurements and is calculated for maximum iterations of 30.

The speed-up of each grouping option against the normal (sequential) LDPC decoding application is measured using the ratio:

$$SpeedUp = TimingSequential/TimingParallel$$

HeMPS performs a System-C based simulation for the configured MPSoC. The output is the decoded bitstream, the number of Trials and the elapsed clock cycles.

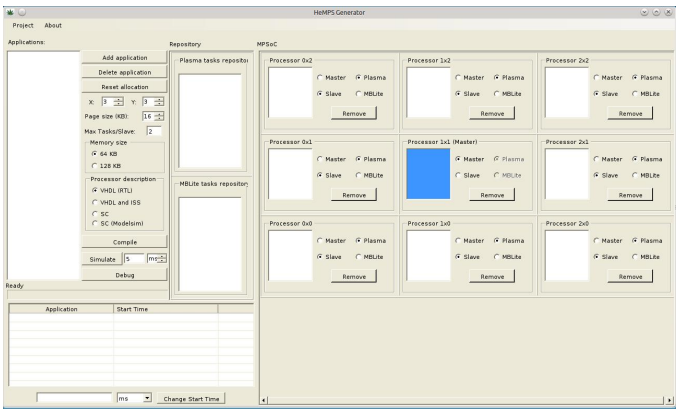

Fig. 7. The HeMPS Simulator

V. Results and Conclusion

The results obtained for MPI framework run on desktop machine and HeMPS simulation are summarised in fig.8 and

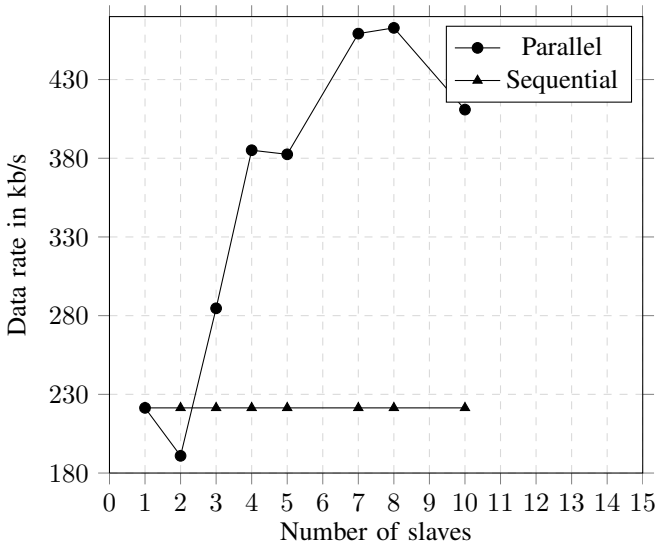

Fig. 8. Throughput rates for sequential and parallel execution of LDPC decoder using MPI framework

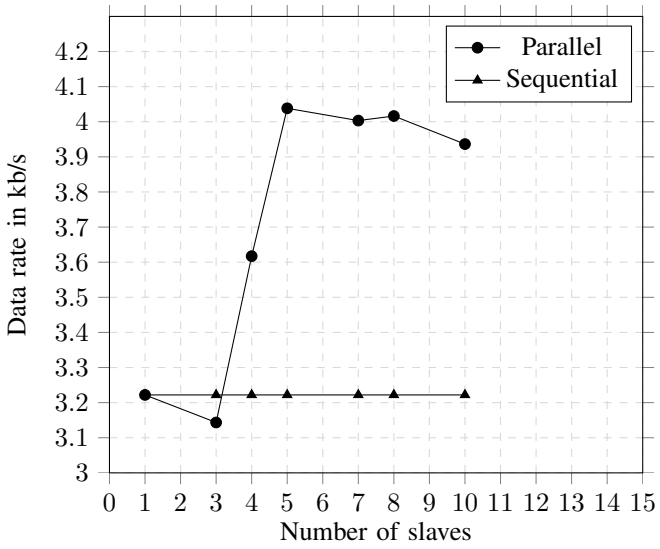

Fig. 9. Throughput rates for sequential and parallel execution of LDPC decoder on HeMPS platform

fig.9.

Seven different scenarios have been considered for measurement were considered. All the scenarios run 1 task per PE only. As the number of check nodes needs to be divisible by the processors, certain processor configurations could not be measured.

1) sequential LDPC decoding application: 1 task, single PE
2) 3-PE allocation (1 master-2 slaves)
3) 4-PE allocation (1 master-3 slaves)
4) 5-PE allocation (1 master-4 slaves)
5) 7-PE allocation (1 master-6 slaves)
6) 8-PE allocation (1 master-7 slaves)
7) 10-PE allocation (1 master-9 slaves)

Results obtained from the simulation are shown in the

following in table I and II.

Scenario	number of cores	Throughput (KBps)	Speed Up factor
1	1	3.222	-
2	3	3.14	0.97
3	4	3.61	1.12
4	5	4.03	1.25
5	7	4.00	1.24
6	8	4.03	1.24
7	10	3.93	1.22

TABLE I
THROUGHPUT AND SPEED UP FACTOR FOR SIMULATION OF LDPC DECODER ON HEMPS

Scenario	number of cores	Throughput (KBps)	Speed Up factor
1	1	221.40	-
2	2	190.93	0.86
3	3	284.67	1.28
4	4	385.10	1.73
5	5	382.51	1.72
6	7	459.19	2.07
7	8	462.83	2.09
8	10	410.90	1.85

TABLE II
THROUGHPUT AND SPEED UP FACTOR FOR SIMULATION OF LDPC DECODER USING MPI FRAMEWORK ON DESKTOP

From the results, for 252x504 matrix, Two important conclusions can be drawn from the results. It can be seen that MPI framework for the current mapping method does not scale up beyond 2. And the scaling factor of HeMPS is lower than that of between MPI framework. In our mapping technique, only check nodes computation was mapped to the slave processors, while a bulk of the processing including variable node processing is performed in the master node. Hence master node forms a bottle neck, in terms of computation and as well as in terms of communication. In addition, HeMPS framework limits the message passing size to only 128 bytes. This adds up to the communication overheads, causing reduced scale-up factor.

As a future work, parallelism can be introduced in variable node update stage as well to see the impact of performance. Increase in the throughput can be achieved by running several such implementations in parallel.

REFERENCES

[1] R. Gallager. Low-density parity-check codes. *Information Theory, IRE Transactions on*, 8(1):21 –28, jan. 1962.

[2] D.J.C. MacKay. Good error-correcting codes based on very sparse matrices. In *Information Theory. 1997. Proceedings., 1997 IEEE International Symposium on*, page 113, jun. 1997.

[3] K.S. Andrews, D. Divsalar, S. Dolinar, J. Hamkins, C.R. Jones, and F. Pollara. The development of turbo and ldpc codes for deep-space applications. *Proceedings of the IEEE*, 95(11):2142 –2156, nov. 2007.

[4] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *Information Theory, IEEE Transactions on*, 24(3):384 – 386, may 1978.

[5] John Dielissen, Andries Hekstra, and Vincent Berg. Low cost ldpc decoder for dvb-s2. In *Proceedings of the conference on Design, automation and test in Europe: Designers' forum*, DATE '06, pages 130–135, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[6] F. Kienle, T. Brack, and N. Wehn. A Synthesizable IP Core for DVB-S2 LDPC Code Decoding. In *Design, Automation and Test in Europe*, pages 100–105. Ieee, 2005.

[7] M. Gomes, G. Falcao, V. Silva, V. Ferreira, A. Sengo, and M. Falcao. Flexible parallel architecture for dvb-s2 ldpc decoders. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 3265 –3269, nov. 2007.

[8] Stefan Grönroos, Kristian Nybom, and Jerker Björkqvist. Complexity analysis of software defined DVB-T2 physical layer. *Analog Integrated Circuits and Signal Processing*, 69(2-3):131–142, 2011.

[9] E.W. Wachter, A. Biazi, and F.G. Moraes. Hemps-s: A homogeneous noc-based mpsocs framework prototyped in fpgas. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, pages 1–8, June 2011.

[10] E.A. Carara, R.P. de Oliveira, N. L V Calazans, and F.G. Moraes. Hemps - a framework for noc-based mpsoc generation. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 1345–1348, May 2009.

[11] A. Anastasopoulos. A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution. In *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, volume 2, pages 1021 –1025 vol.2, 2001.

[12] Xiaofei Huang, Suquan Ding, Zhixing Yang, and Youshou Wu. Fast min-sum algorithms for decoding of ldpc over gf(q). In *Information Theory Workshop, 2006. ITW '06 Chengdu. IEEE*, pages 96 –99, oct. 2006.

[13] IBM Zurich research labs Xiao-Yu Hu. Progressive edge growth construction regular code with rate 1/2. http://www.inference.phy.cam.ac.uk/mackay/codes/data.html#l62.