Mobile Information Systems 8 (2012) 45–59
DOI 10.3233/MIS-2012-0130
IOS Press

Reconfigurable and parallelized network coding decoder for VANETs

Sunwoo Kim[a] and Won W. Ro[b,*]
[a]*Electronics Platform Development Team, Hyundai Motor Company, Republic of Korea*
[b]*School of Electrical and Electronic Engineering, Yonsei University, Seoul, Republic of Korea*

Abstract. Network coding is a promising technique for data communications in wired and wireless networks. However, it places an additional computing overhead on the receiving node in exchange for the improved bandwidth. This paper proposes an FPGA-based reconfigurable and parallelized network coding decoder for embedded systems especially for vehicular ad hoc networks. In our design, rapid decoding process can be achieved by exploiting parallelism in the coefficient vector operations. The proposed decoder is implemented by using a modern Xilinx Virtex-5 device and its performance is evaluated considering the performance of the software decoding on various embedded processors. The performance on four different sizes of the coefficient matrix is measured and the decoding throughput of 18.3 Mbps for the size 16 $\times$ 16 and 6.5 Mbps for 128 $\times$ 128 has been achieved at the operating frequency of 64.5 MHz. Compared to the recent TEGRA 250 processor, the result obtained with128 $\times$ 128 coefficient matrix reaches up to 5.06 in terms of speedup.

Keywords: Network coding, hardware accelerator, VANETs, FPGA

1. Introduction

Data transmission across various Internet connected devices becomes more popular and attractive. One of such applications, inter-vehicular communication draws a significant attention not only for the safety of passengers but also for the content distribution of multimedia services [5,23,31,32]. In fact, the content distribution across moving vehicles is about to be implemented in practice and a sort of infotainment services is already available (or at least will be available shortly) in vehicles such as GENIVI's MeeGo platform [6] and Hyundai's Mozen service [12]. However, due to the high mobility nature of vehicular ad hoc networks (VANETs), implementing sufficient quality of content distribution is still a challenging task. This is due to the fact that network topology changes rapidly and the connections between nodes are not permanently sustained. As a result, the conventional data client model in the traditional network systems is not suitable for VANETs.

To overcome the problems caused with the irregularity of VANETs, network coding can be applied; it helps to achieve maximum bandwidth with exploiting the maximum capacity of given network resources. Especially, it exploits the *broadcast* nature of wireless medium and helps to achieve fast and arbitrary data distribution in VANETs. In fact, it has been shown in recent researches that the poor conditions of VANETs such as mobility, interference, and unreliable channel characteristics can be effectively handled

*Corresponding author: Won W. Ro, School of Electrical and Electronic Engineering, Yonsei University, 262 Seongsanno, Seodaemun-gu Seoul 120-749, Seoul, Republic of Korea. Tel.: +82 2 2123 5769; Fax: +82 2 313 287; E-mail: wro@yonsei.ac.kr.

with employing network coding techniques [20,29]. In network coding based systems, the messages are encoded and forwarded at the intermediate nodes between senders and receivers, instead of just relaying messages as in the conventional network systems.

In order to use the network coding technique in practical VANETs, the long decoding delay at the receiving vehicle should not be the performance bottleneck. In fact, the decoding operation contains many computation demanding procedures such as *Galois Field* operations and large-sized coefficient matrix manipulation. It is a well known fact that the performance degradation due to the computation overhead is not negligible [34]. Due to this decoding complexity at the receiving node, utilizing the network coding technique in practice may not be beneficial as expected.

It has been shown by Lee et al. in [21] that the communication overhead increases as the size of the data block becomes smaller. In addition, the overhead proportionally increases as the size of the original data file becomes large. Also, using blocks in a small size consequently increases the total number of blocks to be transferred. On the other hand, blocks in a large size impose additional computation overhead since the amount of computation is roughly proportional to the size of a block. Therefore, the size of a block should be carefully determined with considering the file size and computation overhead. Otherwise, the computation delay would offset the advantage of network coding.

There have been researches on reducing the computation overhead in network coding [22,25], however, our approach is to use a special decoding hardware accelerator to overcome the problem caused by the long decoding delay. Among various technologies on which the decoder can be implemented, we choose FPGA for the following reasons:

- FPGA has high-reconfigurable nature which makes it easy to provide different types of decoder logic. It gives reconfigurability and flexibility to the design.
- FPGA is suitable for embedded systems which require a certain level of computing capability with low power consumption.

Although embedded processors such as ARM processors can provide a certain level of programmability and flexibility, the computation power of those processors is not sufficient. On the other hand, computation power is not a problem for the conventional general processors; the problem is that this sort of processors requires a complete size computer system which is far more complex and larger than embedded systems. In addition, the complete system introduces more serious problem with significant power consumption. Therefore, considering the vehicular environment, conventional processors are not recommended for practical use. In fact, several recent researches have shown the feasible software implementations with the environment of conventional processors [7,28,34].

In this paper, we propose a special decoding logic design which has a parallelized architecture for high-performance network coding with the help of the FPGA technology. Our contribution in this paper is to provide a design which can manage excessive calculations and can enhance the decoding speed. In fact, the majority of the decoding process is related to the coefficient vector operations and our main contribution can be found in expediting the coefficient vector operations with Galois Field numbers. The proposed design which aggressively exploits the parallelism inside network coding accelerates the arithmetic operations and reduces the number of computations for high-performance decoding procedure.

One of the main advantages using the FPGA technology is that the network coding decoder would require reconfigurability due to the change of protocols such as the number of coefficient vectors and block size. Therefore, using a hardware-based accelerator with ASIC might not be a good solution. Truly, the proposed design and implementation give fundamental advantages considering the trade-offs between the software oriented approach and the fixed hardware design.

Throughout this research, various decoders considering four different sizes from 16 × 16 to 128 × 128 are implemented on an FPGA. With the FPGA technology, a great level of flexibility is maintained while the high performance is achieved with exploiting parallelism in each decoder. From our FPGA implementation, we achieve the decoding throughput of maximum of 18.3 Mbps. The performance is evaluated by considering the performance in various embedded processors. The speedup factor we achieve is 5.1 compared to the results obtained from the recent Cortex-A9 processor, TEGRA 250 (dual-core processor).

The rest of paper is organized as followings. Background researches and related works are covered in Section 2. In Section 3, the implementation of the decoder including the descriptions on the decoding process flow is given. The proposed design of network coding accelerator and its parallelized architecture are also described in this section. In Section 4, the simulation results are presented along with the detailed performance analysis. Finally, this paper is concluded in Section 5.

2. Background research and related work

The major contribution of the paper is to provide a network coding accelerator which focuses on the two major computation demanding parts: operations on the coefficient vectors and computations on Galois Field numbers. For that purpose, we optimize the decoding algorithm by developing a parallelized hardware structure and special logic which are customized for the network coding procedure. In addition, fast multiplication and division operations are achieved by the help of the log and anti-log tables. All data elements in the network coding are Galois Field numbers and the multiplications and divisions for decoding operations have been identified as the most complicated and time-consuming tasks [11].

2.1. Issues on designing network coding accelerators on VANETS

In this research, the random linear coding [4,10] has been chosen as a target coding method; it is known to be asymptotically optimal in any types of networks. When we design a detailed hardware structure for the random linear coding, a novel design is required to avoid the time-consuming decoding operations. In addition, the frequent topology changes should be addressed in the VANET environment. This additional feature might introduce necessary modification in the protocol aspect of network coding such as the number of blocks in a generation or information on coefficient vectors. In fact, the coding bandwidth varies with the number of blocks for the fixed block size, and is generally determined by the network capacity and computing power [39]. To efficiently cope with the frequent changes, a certain level of programmability needs to be provided.

Under the system with a general-purpose programmable processor, the number of blocks needs not to be fixed since modification of an algorithm written in a high-level language is relatively easy. However, using commercial general-purpose processors introduces another problem in terms of decoding performance. Software-oriented decoding method on a general purpose CPU does not provide fast enough decoding speed and additionally requires a full computer system which is far more complex and larger than a customized embedded system. Our proposed system eventually targets to be used in VANETs, as an integrated embedded system in a vehicle.

To develop a customized decoder in hardware with sufficient reconfigurability, we have chosen the FPGA technology. In fact, the field programmable gate-array (FPGA) technology gives a solution to this problem as it is widely used in the various fields such as network systems [27] and cryptography [15, 40]. By virtue of the FPGA's high-reconfigurable nature, the modification of hardware design can easily

be achieved. In this paper, we propose a decoder, designed based on reconfigurable core logic, for four different sizes of coefficient vectors. It only needs little change of the core logic and the bus interface to reconfigure the decoder for a certain number of blocks.

2.2. Related works

The fundamental concept of network coding has first been introduced by Yeung et al. [39] for satellite communication networks. The concept has been fully developed in a subsequent research by Ahlswede et al. [1]. Since introduced, it has received significant attention and various researches have followed. Pederson et al. [30] have implemented and evaluated XOR-only network coding.

In fact, random linear network coding is chosen in most of recent researches such as Chachulski et al. [3] and Katti et al. [16]. Random linear network coding uses Galois Field numbers for the coding symbols, on which the arithmetic operations demand heavy computation overhead. Several algorithms and architectures such as LFSR multipliers [19] and Mastrovito multipliers [24] have been proposed for the arithmetic operations over Galois Field. Particularly, there have been numerous investigations on faster and efficient arithmetic over Galois Field on FPGAs motivated by various areas such as data encryptions. Kerins et al. [17] have studied algorithms and architectures using FPGAs with the field of GF(3^m). Spagnol et al. [33] proposed a hardware implementation of serial architecture for LDPC over GF(2^m). However, the increment of complexity in hardware design has always been accompanied as shown in their work.

Besides, many researchers have taken another approach to improve the performance of random linear network coding; most of the researches have been conducted on conventional desktop environment. Shojania and Li have proposed progressive network coding method in [34]. The decoding performance has been improved with progressive decoding as the decoding time has been overlapped with the time required for data transmission. Park et al. have demonstrated that the strategy of parallelization in network coding gives significant impact to the decoding performance [28]. They have emphasized *balanced* parallelization for both the decoding performance and the scalability of the number of processors. Those features are very important in parallel processing [36]. On the other hand, there has been a research on implementing random network coding on embedded systems. Shojania and Li [35] have implemented random network coding on hand-held devices in their recent research. iPhone and iPod Touch have been chosen for the devices and the performance of the ARM processors used in those devices has also been evaluated.

Meanwhile, several researches have been conducted on adapting network coding in vehicular environment. The key technology that enables the vehicular communications is Dedicated Short Range Communication (DSRC) [14] which allocates channels for both safety and non-safety purpose. Nguyen et al. have introduced network coding schemes to reduce the number of broadcast transmissions [26]. With the schemes, multiple receivers collect and combine the lost packets even from another receivers and recover their lost packets. In this manner, the number of retransmission by the sender due to the packet loss can be reduced. Ahmed and Kanhere have proposed a network coding based cooperative content distribution scheme called VANETCODE [2]. Lee et al. have investigated on the overheads of network coding in VANETs [21]. Their research has shown that the resource constraints such as disk I/O and computation overhead have a significant impact on the performance.

To the best of our knowledge, our work is the first research work to propose network coding accelerators on FPGAs. There has been a research about adopting network coding to FPGA technology [8], however their contribution is to improve the FPGA's *internal routing characteristics* by applying the technique of network coding. Consequently, the research is strictly focused on improving routing capability within basic logic blocks in FPGA.

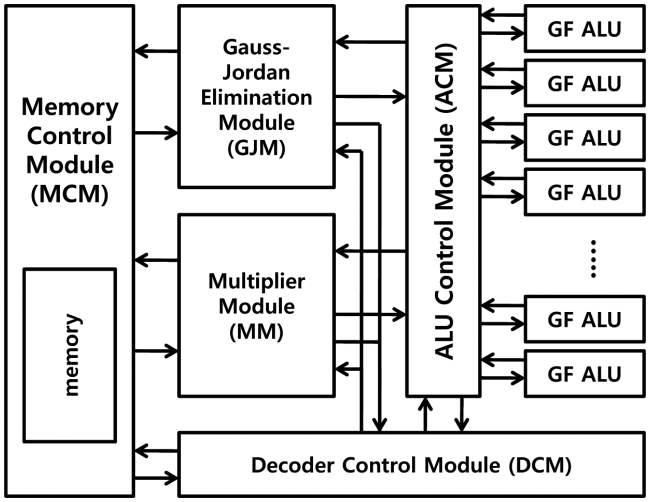


Fig. 1. Block diagram of the decoder.

3. Decoder implementation on FPGA

In this section, we describe a detailed description of the proposed design. Inside the decoder, we have incorporated five functional modules. Among them, the three main functional modules are Decoder Control Module (DCM), Gauss-Jordan Elimination Module (GJM), and Multiplier Module (MM). In general, the decoding process is performed by GJM and MM under the control of DCM. These modules share the ALUs over the entire decoding process in order to reduce the area overhead. In our design, we name the ALU structure as Galois Field ALUs (GF ALUs).

3.1. Description of the proposed decoder on FPGA

The detailed block diagram of the proposed decoder design is given in Fig. 1. DCM accepts the encoded data from outside of the decoder module and stores the data to the memory in the Memory Control Module (MCM). GJM and MM only accept the execution trigger signals from the main control module and begin their operations. In order to reduce the area overhead, ALU Control Module (ACM) is connected to both GJM and MM. These two modules share the same ALU logic and the main control module gives a select signal to specify the module which is allowed to use the GF ALUs at a given time.

To describe the detailed working flow of the proposed design, we prepare task descriptions for the three stages on Table 1. In the first stage, the main control module collects the encoded data from the buffer on a network device. In fact, data to be decoded is only allowed to be received by the decoding logic when the decoder is not in the busy state. To avoid the data incoming during the decoding process, the main control module has a signal to inform the status of the decoding logic. At the receiving the signal from the decoder, the outside buffer starts to send the coded packets to the decoder.

The encoded data from the outside buffer is received until a predetermined number of independent data frames are stored in the memory. After that, the decoder module should assure that all the data frames are independent. Otherwise, the decoder cannot obtain the inverse matrix of coefficient matrix due to the insufficient rank of the matrix. As soon as all the necessary coded packets are received, the main control module controls the logic to move on to the next stage by sending the trigger signal to the GJM.

Table 1
Utilization of the components

Stage	Task descriptions
1	Main controller accepts the encoded data from outside. The collected data is stored in memory.
2	Gauss-Jordan elimination is performed on the coefficient matrix by GJM.
3	MM restores the original data by using the inverse matrix of the coefficient matrix created in Stage 2.

At the second stage, the GJM mainly performs the Gauss-Jordan operations on the coefficient matrix and obtains the inverse matrix of it. For the Gauss-Jordan operations, the GJM has two buffers in itself – named RC to hold the coefficient matrix and named RI to hold the inverse matrix. Both buffers can hold as large a matrix as coefficient vectors. At the beginning of the stage, the identity matrix is stored in RI while the coefficient vectors are loaded from the memory to RC. Once the initialization is finished, the GJM starts the operations required for the matrix inversion which exploit parallelism considering size of the coefficient vectors.

The Multiplier Module (MM) takes its part in the last stage of the decoding process. Similar to GJM, MM also has its own buffer to store the inverse matrix of the coefficient vectors and thus the size of buffer is same as the size of the inverse matrix. Once a control signal from the main control module triggers MM, the module loads the inverse matrix from the memory and holds it in the buffer.

Finally, the last stage of the decoding procedure is processed. Each column of the data matrix in the memory is loaded and stored in a buffer in parallel. The column data in the buffer and a row of inverse matrix in the buffer are then multiplied. The results of the operation are XORed and the original data is finally restored. While the multiplication is in progress, MM loads the next column data from the memory in order to prepare the subsequent operation.

As mentioned above, we introduce a parallelized decoding logic for high-performance network coding. For that purpose, the array of GF ALUs are proposed and used for the most of the calculations which are related to the coefficient vectors and data; we can easily achieve simultaneous execution of multiple multiplications or divisions. In order to exploit parallelism on a single row, the number of ALUs in the array of GF ALUs is designed to match the number of elements in a coefficient vector and, in consequence, we can simultaneously execute all the required operations for any row or column operation.

3.2. Galois field arithmetic logic unit

In the Galois Field operation, the addition and subtraction are equivalent; both can be performed by bit-wise XOR operations. However, the multiplication and division are more complex and computationally expensive tasks. In general, these operations are performed based on extended Euclidean algorithm (EEA) or Fermat's little theorem. Many architectures based on EEA or Fermat's little theorem exist, e.g. [13,18,38], however they are usually expensive in terms of area. In our design, however, we take an algorithm-independent approach using log and anti-log tables to provide a simple design which is suitable for the embedded systems.

For our design, we have started from the fact that a multiplication of two numbers is represented as an addition when we take log of those numbers. Similarly, a division can be represented as a subtraction. This observation implies that we can substitute addition and subtraction for multiplication and division if we take anti-log of the log-taken Galois Field numbers. In other words, multiplication and division operations over Galois Field can be simply achieved by accessing the log table for each operand, adding or subtracting the retrieved log numbers, and finally another access to the anti-log table. Circuit complexity and calculation time can be reduced in multiplication and division by taking the log of Galois Field numbers.

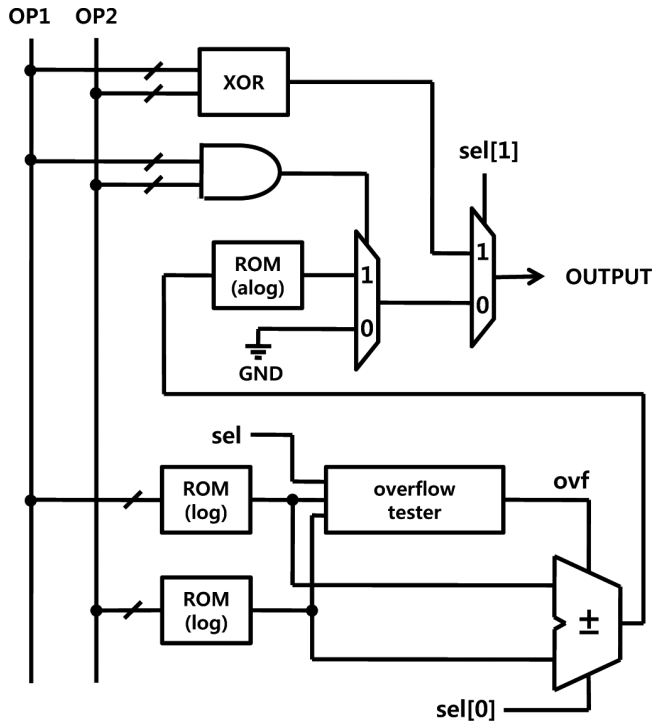


Fig. 2. Connection of GF ALU and neighboring modules.

The process of taking log and anti-log may seem to be an additional overhead for the procedure. However, this problem can be easily solved by using ROMs in an FPGA chip; the ROMs can contain the log and anti-log tables. Since we use the Galois Field numbers, the size of the tables is bounded to the size of the set. In our design, each ALU simply can contain three ROMs which provide the information of log and anti-log tables, exploiting complete parallelism over a coefficient vector by executing the dependency-free operations simultaneously. With this approach, we can obtain suitable performance for the arithmetic operations without using complex algorithm or implementation.

The structure of GF ALU and supporting components are depicted in Fig. 2. Each GF ALU module has dedicated ROMs for two log tables, another ROM for an anti-log table, and an overflow tester. The Galois Field numbers in 8-bit representation are converted by referring the two log tables with their corresponding log numbers. On the other hand, the anti-log table is used for the reverse conversion. The overflow tester is designed to perform modulo operations.

Input signals to the GF ALU logic are two 8-bit operands from the ALU Control Module with a 2-bit ALU select signal. The proposed ALU performs arithmetic operations of multiplication, division, or addition/subtraction on these two operands in Galois Field and the select signal indicates which type of operation the ALU should execute. Before concluding the multiplication/division operation, it is required to check whether any of the two operands is zero or not. If so, the output should be zero instead of the anti-log result and the mux on the left decides the result.

The behavior of GF ALU operation is given in a form of the pseudo code in Fig. 3. The ALU executes an operation according to the select signal. The ALU operation is exclusive OR if the select signal is either addition or subtraction and is represented in the pseudo code using symbol $\oplus$. In the case of multiplication and division, although the multiplication and division operations are replaced by the

```
Input : two operands op1 and op2
Output : ALU operation result
switch(operation select signal)
  case: addition or subtraction
    result=op1⊕op2;
    break;
  case: multiplication
    result=log[op1] + log[op2];
    result=result mod 255;
    result=anti-log[result];
    if(op1=0 or op2=0) result=0;
    break;
  case: division
    result=log[op1] - log[op2];
    result=result mod 255;
    result=anti-log[result];
    if(op1=0 or op2=0) result=0;
    break;
// end of switch statement
return result;
```

Fig. 3. Pseudo code of GF arithmetic logic unit.

addition and subtraction operations by taking log and anti-log, it is still required to handle the overflow of addition and subtraction operations as represented in the pseudo code.

In the code, the overflow is handled by the modulo operation of the largest value in the set, which is 2^8-1 in the case of GF(2^8). However, this is not a practical method when it is implemented with Verilog HDL since only modulo operation of 2^n is allowed. Modulo of any 2^n numbers are implemented as taking lower n bits by the compiler, however typical compiler tool does not support modulo operation of any other numbers. Therefore, an overflow tester is implemented to avoid this problem. From the tester, the overflow signal (represented as *ovf* in Fig. 3) is sent to the adder/subtractor to inform the occurrence of the overflow and subsequently, the adder/subtractor adjusts output of the calculation.

While the ALU executes the arithmetic operation of the log-converted operands, it always sends the operands to the overflow tester and checks whether there is an overflow in the results. The delay of the tester is short enough not to be a bottleneck. Finally, GF ALU checks if any of the operands is zero or not and returns the appropriate result.

3.3. ALU control module

As described briefly in Section 3.1, we propose a parallelized ALU architecture using log tables to enhance the calculation speed. These ALUs are connected to the ALU Control Module (ACM) through the bit distributers. Each GF ALU operates independently from other ALUs and thus the ALUs can simultaneously execute 16, 32, 64, or 128 GF arithmetic operations, depending on the size of the coefficient vector. Figure 4 depicts the structure of ACM and GF ALUs.

In our design, the number of GF ALUs is strictly limited to the size of the coefficient matrix; the increasing number of GF ALUs introduces a significant area overhead. GJM and MM share the ALUs and the ACM module basically works as a multiplexing logic. To recognize the valid input, the SELECT signal is given according to the decoding stage.

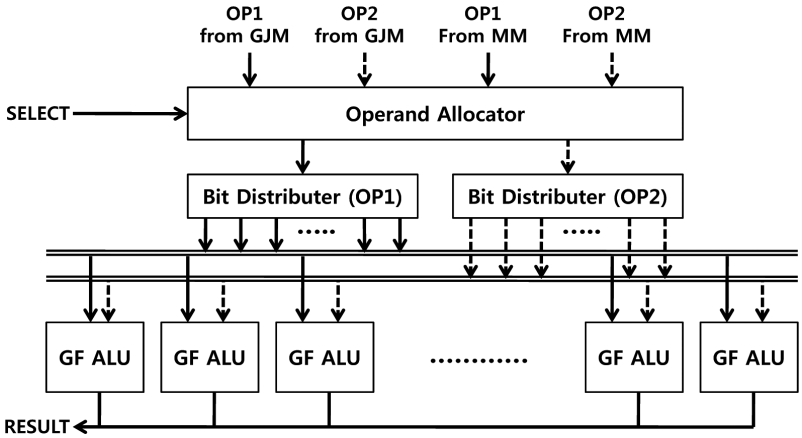


Fig. 4. Connection of ALU control module and GF ALUs.

The ACM accepts two operands as inputs from both GJM and MM, and the SELECT signal from the main control module. Afterwards, the ACM checks the SELECT signal and decides which operands to use for the calculation. The selected operands are then sent to each bit distributer which parses the input signals into 8-bit operands and distributes them to proper GF ALUs respectively. After the arithmetic operation is performed by the GF ALUs, the results are merged again to be sent back to the ACM. Meanwhile, the main control module maintains its state until the calculation is finished and guarantees the ACM to send the results back to either the GJM or the MM properly. Once the results are transferred to those modules, the main controller continues the decoding process and the ACM performs the next calculations.

4. Performance evaluation and analysis

As described briefly earlier, the proposed design is implemented on a Xilinx FPGA device. The performance is compared to the various ARM processors.

4.1. Experimental methodology

The proposed design is synthesized by Xilinx ISE 10.1 and XC5VLX110T is chosen for the target device, which is one of the middle-end devices in the Virtex-5 family. The device provides 17,280 Virtex-5 slices and maximum 5,328-Kbits BlockRAM blocks [37]. The module is evaluated by performing post place and route simulation on ModelSim 6.3f.

To evaluate the performance of our design, we have measured performance of the software approach on various ARM processors. The ARM processors used in evaluation include ARM922T, ARM926EJ-S, ARM1176JZF-S, and Cortex-A9; the chosen devices are categorized into architecture versions and each model is one of the most commercially successful architectures. The main differences are the size and structure of cache, pipeline stages, maximum operating frequency, etc. For ARM1176JZF-S, we have used RealView Developer Suite (RVDS) v4.0. The suite program offers a debugger which can emulate most of the ARM processor cores including the ARM11 series cores and supports diverse internal information such as the contents of memory, register file, and the count of core cycles and clocks. The code is cross-compiled by arm-linux-gcc with -O3 optimization level and also executed on

Table 2
Utilization of the components

Size of the coefficient matrix	Slice flip-flop (%)	Slice LUTs (%)	Block RAM (%)
16 × 16	1	6	8
32 × 32	2	12	15
64 × 64	4	27	31
128 × 128	9	57	62

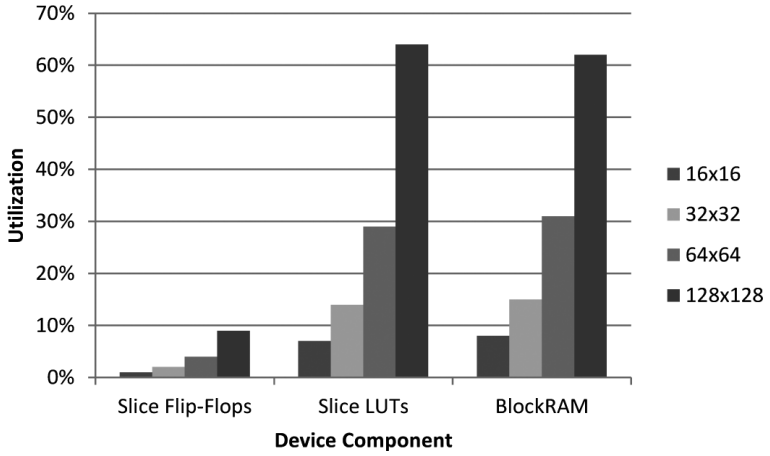


Fig. 5. Utilization of FPGA design on XC5VLX110T.

Linux environment. In addition to the ARM processor models mentioned above, we have tested Octeon CN5320 network processor.

4.2. Synthesis results

Xilinx FPGA devices are composed of several functional blocks. Among them, utilization on the Configurable Logic Blocks (CLBs) and BlockRAM blocks is important since our design is mainly implemented on these structures. In fact, a CLB contains a group of slices; a slice contains four look up tables (LUTs) and four flip-flops in the case of Virtex-5 family [37].

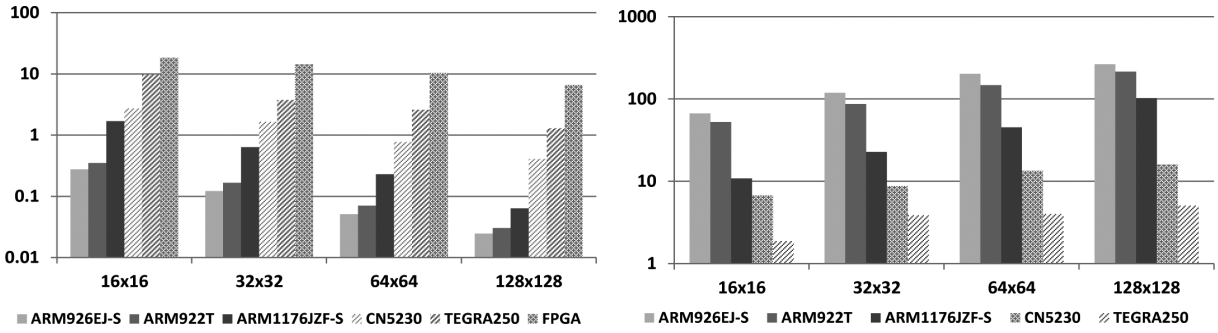
The utilization of the components for each size of the coefficient matrix is given in Table 2 and their graphical comparison is depicted in Fig. 5. The results show the feasibility of our design for the proposed four sizes. The utilization of Slice Flip-Flops is 1% while the Slice LUTs and BlockRAM show 6% and 8% for the coefficient matrix size of 16 × 16. The utilization relatively increases as the matrix size doubles, reaching up to 9%, 57%, and 62% for the size of 128 × 128. Hence, 128 × 128 is the maximum size of the coefficient vector fitting to the chosen device size.

4.3. Simulation results and analysis

Through the decoding process of the network coding, the most important factor that limits the performance is related to the matrix operation. Therefore, we focus on the structure of the GF ALUs to exploit the parallelism for the maximum throughput. As previously mentioned, we have measured the decoding time on real ARM922T, ARM926EJ-S, and recent Cortex-A9 processors for the comparison. We also have performed the experiment on the Octeon CN5230 network processor. ARM922T is an

Table 3
List of the processors used in simulation

	ARM922T	ARM926EJ-S	ARM1176JZF-S	CN5230	TEGRA250
Base Architecture	ARMv4T	ARMv5TEJ	ARMv6KZ	cnMIPS64	ARMv7-A
Operating Frequency	133 MHz	100 MHz	800 MHz	700 MHz	1GHz
Simulation Type	Real Chip	Real Chip	RVDS	Real Chip	Real Chip



(a) (b)

Fig. 6. (a) Throughput of FPGA implementation and other processors; (b) Speedup of FPGA implementation compared with the simulation results on other processors.

FPGA-implemented processor using Altera Excalibur working at the operating frequency of 133 MHz, and ARM926EJ-S is an ASIC chip using 0.18 um technology working at the operating frequency of 100 MHz. Both belong to the ARM9 processor group, however the architecture version of them are ARMv4 and ARMv5, correspondingly. Among the ARM-based processors, Cortex-A9 is the most recent and powerful processor with a speculative issue of instructions and superscalar execution. For Cortex-A9 processor experiment, NVIDIA's TEGRA250 is chosen for the device. It belongs to the architecture version of ARMv7 and is the second commercial Cortex-A9 processor to be announced. Based on TSMC's 40nm process technology, TEGRA250 works at the frequency of 1 GHz. Octeon CN5230 is a network processor of CAVIUM Networks, and is a MIPS64-based SoC processor. The working frequency in the experiment is 700 MHz.

While TEGRA 250 represents the top edge of ARM-based processors, ARM926EJS represents the most successful ARM architecture in commercial area. However, as we can see from their architecture versions, the performance gap of those processors is substantial. For this reason, we additionally have emulated ARM1176JZF-S to test the same algorithm on RVDS. ARM1176JZF-S is a successful architecture in ARMv6 family and thus is chosen for the representative. To measure the performance of the processor on RVDS, we have counted the core clock cycles for each case of the coefficient matrix size 16×16, 32×32, 64×64 and 128×128. The clock count is then converted to the throughput considering the operating frequency; frequency is chosen to 800 MHz according to the specification of S3C6410 which is an ARM1176JZF-S processor produced by Samsung. The processors used in simulation are listed with their features and operating frequencies in Table 3.

For our FPGA design, the results of post place and route implementation give maximum achievable working frequency of 64.5 MHz, and thus is the choice for the simulation. The results of the simulation are depicted in Fig. 6. Note that Y-axis is in log scale in both graphs.

The results of the simulation are depicted in Fig. 6-(a). The results show that the proposed design gives the best performance. The FPGA implementation gives 18.324 Mbps on the size of 16×16 while

Table 4
Throughput of each processor

Matrix Size	ARM 926EJ-S	ARM 922T	ARM 1176JZF-S	Octeon CN5230	TEGRA250	FPGA
16 × 16	0.275	0.350	1.688	2.718	9.766	18.324
32 × 32	0.121	0.166	0.634	1.645	3.720	14.397
64 × 64	0.051	0.070	0.228	0.770	2.583	10.337
128 × 128	0.025	0.030	0.064	0.407	1.290	6.528

ARM926EJS, ARM922T, ARM1176JZF-S, CN5230, and TEGRA250 processors give 0.275 Mbps, 0.350 Mbps, 1.688 Mbps, 2.718 Mbps and 9.766 Mbps respectively. In the case of the size 128 × 128, 6.528 Mbps of the decoding throughput is recorded by the FPGA implementation whereas the results of 0.025 Mbps, 0.030 Mbps, 0.064 Mbps, 0.407 Mbps and 1.290 Mbps are recorded by the other processors. In the case of ARM9 processors, we can observe that the results of ARM922T show better performance than the results of ARM926EJ-S. Generally, the latter architecture is considered as more powerful and refined, and consequently, shows better performance than the former one. However the results can be reversed by the difference of working condition. In our simulation, the working frequency of ARM922T is 133 MHz, which is 33% higher than ARM926EJ-S and it results in the higher performance of ARM922T.

The most important factor that improves the performance of our design is the simultaneously executed one-row-operation by the parallelized GF ALUs. During Gauss-Jordan elimination, the number of operations on each pivot element $(i,\ i)$ is $n(2n+1-i)$ as the algorithm changes the coefficient matrix into an identity matrix while an identity matrix of the same size is changed into the inversed coefficient matrix. Here we count one multiplication and one subtraction as one operation. Therefore, the total number of the operations can be calculated as Eq. (1) which can be regarded as the execution time for Gauss-Jordan elimination.

$$\sum_{i=1}^{n} n(2n+1-i) = 2n^3 + n^2 - \frac{n^2(n+1)}{2} \tag{1}$$

Regarding the proposed design with multiple GF ALUs, however, the number of operations on each pivot element is $2(2n-1)$ since the one-row-operation is executed simultaneously. Therefore, the total number of the operations is $2n(2n-1)$. In this case, we count both multiplication and subtraction with the same weight as they execute in the same number of clock cycles in the GF ALU structure. By comparing this operation count with Eq. (1), we can get the result as in Eq. (2). In the equation, the order of the numerator is higher than the denominator which allows us to deduce that the proposed design would achieve higher improvement of the performance as the size of the coefficient vector gets larger.

$$\frac{2n^3 + n^2 - \frac{n^2(n+1)}{2}}{2n(2n-1)} = \frac{3n^2 + n}{4(2n-1)} \tag{2}$$

Figure 6-(b) shows the speedup of FPGA design comparing with the simulation results on the other processors. For the coefficient matrix size of 16 × 16, the maximum speedup is recorded 66.68 on ARM926EJ-S. With the coefficient matrix size of 128 × 128, the speedup factor of as much as 264.80 is achieved compared to ARM926EJ-S. More realistically, the speedup comparing with TEGRA 250 records 5.06 for the size 128. It degrades to 1.88 as the size shrinks to 16 however still it means 88% of performance improvements.

As a result, the proposed design shows better performance for the larger size of the coefficient matrix which is the same result of our deduction. The throughput of each size of the coefficient matrix on each processor is given in Table 4 and the corresponding speedup is given in Table 5.

Table 5
Speedup of FPGA design

Matrix Size	ARM 926EJ-S	ARM 922T	ARM 1176JZF-S	Octeon CN5230	TEGRA250
16×16	66.68	52.42	10.86	6.74	1.88
32×32	118.74	86.75	22.71	8.75	3.87
64×64	202.79	147.31	45.25	13.42	4.00
128×128	264.80	215.90	102.41	16.04	5.06

5. Conclusion

To overcome the computation overhead which is one of the major obstacles to use network coding in VANETs, a high-performance decoder for the network coding has been proposed with the FPGA technology. In our design, a special ALU for the Galois Field operations with log and anti-log table is implemented for simple and fast arithmetic operations. In addition, a parallelized structure of GF ALUs is developed to maximize throughput. Four different types of the decoder have been implemented for the different sizes of the coefficient vectors, and those have been compared to the performance on ARM processors by emulating on RVDS and executing on real processor devices.

In vehicular environment, there has been a trade-off between the embedded processors and conventional general purpose processors. While the embedded processors suffer from the lack of computation power, conventional processors have problem with the requirements of complex full computer system and lack of architectural flexibility. Using the FPGA technology, the computation power and system complexity is well balanced with FPGA's design flexibility and the requirements of vehicular environment are satisfied.

For the four different sizes of the coefficient matrix, it has been shown that the larger size of the coefficient matrix gives better performance in terms of speedup. All over the simulation and experimental results, it has been proven that the proposed decoder gives much higher performance than the performance of the software decoding operations on the ARM processors. Our proposed design provides the speedup which ranges from 1.88 to 66.68 with the size of 16×16 and from 5.06 to 264.80 with the size of 128×128.

On the extension of this research, we will explore further on larger sizes of coefficient matrix. It has been found that the number of GF ALUs cannot be greater than 128 on the chosen device. This limitation implies that in order to support larger coefficient matrices, modification of the architecture is required. For that reason, we will modify the control scheme to overcome the limitation of the number of GF ALUs. Additionally, we will adopt a pipelined implementation to retrieve the possible degradation of the performance by the modification.

This work has confirmed that the high performance FPGA implementation of decoder logic for network coding is feasible. Due to the heavy calculation overhead of the arithmetic operation over Galois Field and matrix operations, an optimized decoder with highly parallelized ALU structure is a better choice for higher throughput of network coding.

Acknowledgements

This research was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2009-0077326) and in part by the research funding program of Korea Sanhak Foundation (2010).

References

[1] R. Ahlswede, N. Cai, S.R. Li and R.W. Yeung, Network information flow, *IEEE Information Theory* **46**(4) (2000), 1204–1216.

[2] S. Ahmed and S.S. Kanhere, VANETCODE: network coding to enhance cooperative downloading in vehicular Ad-Hoc networks, In Proc. of the International Wireless Communications and Mobile Computing Conference, Vancouver, Canada, 2006, pp. 527–532.

[3] S. Chachulski, M. Jennings, S. Katti and D. Katabi, Trading structure for randomness in wireless opportunistic routing, In Proc. of ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, 2007, pp. 169–180.

[4] P.A. Chou, Y. Wu and K. Jain, Practical network coding, In *Proc. of 41st Annual Allerton Conference on Communication, Control, and Computing*, Monticello, USA, 2003.

[5] T. Delot, S. Ilarri, N. Cenerario and T. Hien, Event sharing in vehicular networks using geographic vectors and maps, using geographic vectors and maps, *Mobile Information Systems* **7**(1) (March 2011), 21–44.

[6] GENIVI: MeeGo platform, http://www.genivi.org.

[7] C. Gkantsidis, J. Miller, P. Rodriguez, Comprehensive view of a live network coding P2P system, In *Proc. of ACM/SIGCOMM Internet Measurement Conference*, Rio de Janeiro, Brasil, 2006, pp. 177–188.

[8] K. Gulati and S.P. Khatri, Improving FPGA routability using network coding, In *Proc. of 18th ACM Great Lakes Symposium on VLSI*, Orlando, USA, 2008, pp. 147–150.

[9] A.H. Ho, Y.H. Ho, K.A. Hua, R. Villafane and H.-C. Chao, An efficient broadcast technique for vehicular networks, *Journal of Information Processing Systems* **7**(2) (June 2011).

[10] T. Ho, M. M'edard, R. Koetter, D.R. Karger, M. Effros, J. Shi, B. Leong, A random linear network coding approach to multicast, *IEEE Information Theory* **52**(10) (2006), 4413–4430.

[11] Y.T. Horng and S.W. Wei, Fast inverters and dividers for finite field GF(2^m), In *Proc. of IEEE Asia-Pacific Conference on Circuits and Systems*, Taipei, Taiwan, 1994, pp. 206–211.

[12] Hyundai: Mozen service, http://www.mozen.com.

[13] T. Itoh and S. Tsujii, A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases, *Information and Computation* **78**(3) (1988), 171–177.

[14] D. Jiang, V. Taliwal, A. Meier, W. Holfelder and R. Herrtwich, Design of 5.9 GHz DSRC-based vehicular safety communication, *IEEE Wireless Communications* **13**(5) (2006), 36–43.

[15] M.-H. Jing, Z.-H. Chen, J.-H. Chen and Y.-H. Chen, Reconfigurable system for high-speed and diversified AES using FPGA, *Microprocessor and Microsystems* **31**(2) (2007), 94–102.

[16] S. Katti, D. Katabi, H. Balakrishnan and M. Medard, Symbol-level network coding for wireless mesh networks, *ACM SIGCOMM Computer Communication Review* **38**(4) (2008), 401–412.

[17] T. Kerins, E. Popovici and W. Marnane, Algorithms and architectures for use in FPGA implementations of identity based encryption schemes, *Lecture Notes in Computer Science* **3203** (2004), 74–83.

[18] C.H. Kim and C.P. Hong, High-speed division architecture for GF(2^m), *Electronics Letters* **38**(15) (2002), 835–836.

[19] S. Lin and D. Costello Jr., Error control coding: fundamentals and applications, Englewood Cliffs, NJ: Prentice Hall, 1983.

[20] U. Lee, J.S. Park, J. Yeh, G. Pau and M. Gerla, CodeTorrent: Content distribution using network coding in VANET, In *Proc. of the 1st International Workshop on Decentralized Resource Sharing in Mobile computing and Networking*, Los Angeles, USA, 2006, pp. 1–5.

[21] S.H. Lee, U. Lee, K.W. Lee and M. Gerla, Content distribution in VANETs using network coding: the effect of disk I/O and processing O/H, In *Proc. of 5th Annual IEEE Communications Society Conference on Sensor*, Mesh and Ad Hoc Communications and Networks, San Francisco, USA, 2008, pp. 117–125.

[22] G. Ma, Y. Xu, M. Lin and Y. Xuan, A content distribution system based on sparse linear network coding, In *Proc. of Third Workshop on Network Coding, Theory, and Applications*, San Diego, USA, 2007.

[23] S.S. Manvi, M.S. Kakkasageri and Jeremy Pitt, Multiagent based information dissemination in vehicular ad hoc networks, *Mobile Information Systems* **5**(4) (December 2009).

[24] E. Mastrovito, VLSI architectures for computation in galois Fields, PhD thesis, Dept. of Electrical Eng., Linkoping Univ., Sweden, 1991.

[25] P. Maymounkov, N.J.A. Harvey and D.S. Lun, Methods for efficient network coding, In Proc. of 44th Annual Allerton Conference on Communication, Control, and Computing, Monticello, USA, 2006.

[26] D. Nguyen, T. Tran, T. Nguyen and B. Bose, Wireless broadcast using network coding, *IEEE Vehicular Technology* **58**(2) (2009), 914–925.

[27] A. Nikologiannis, I. Papaefstathiou, G. Kornaros and C. Kachris, An FPGA-based queue management system for high speed networking devices, *Microprocessor and Microsystems* **28**(5–6), (2004), 223–236.

[28] K. Park, J.S. Park and W.W. Ro, Efficient parallelized network coding for P2P file sharing applications, In *Proc. of 4th International Conference on Grid and Pervasive Computing*, Geneva, Switzerland, 2009, 353–363.
[29] S. Park, M. Gerla, D.S. Lun and Y.Yi, M. Medard, CodeCast: a network-coding-based Ad Hoc multicast protocol, *IEEE Wireless Communications* **13**(5) (2006), 76–81.
[30] M.V. Pedersen, F.H.P. Fitzek and T. Larsen, Implementation and performance evaluation of network coding for cooperative mobile devices, In *Proc. of IEEE International Conference on Communications Workshops*, Beijing, China, 2008, pp. 91–96.
[31] N. Qadri, M. Altaf, M. Fleury and M. Ghanbari, Robust video communication over an urban VANET, *Mobile Information Systems* **6**(3) (August 2010).
[32] M. Raya and J.-P. Hubaux, Securing vehicular ad hoc networks, *Mobile Information Systems* **15**(1) (January 2007).
[33] C. Spagnol, W. Marnane and E. Popovici, FPGA implementations of LDPC over GF(2^m) decoders, In *Proc. of IEEE Workshop on Signal Processing Systems*, Shanghai, China, 2007, pp. 273–278.
[34] H. Shojania and B. Li, Parallelized progressive network coding with hardware acceleration, In *IEEE International Workshop on Quality of Service*, Evanston, USA, 2007, pp. 47–55.
[35] H. Shojania and B. Li, Random network coding on the iPhone: fact or fiction? In *Proc. of 18th international Workshop on Network and Operating Systems Support for Digital Audio and Video, Williamsburg*, Virginia, 2009, pp. 37–42.
[36] D. Taniar, C.H.C. Leung, W. Rahayu and S. Goel, High Performance Parallel Database Processing and Grid Databases, John Wiley & Sons, 2008.
[37] Xilinx, San Jose, CA, Virtex-5 family overview, 2006, available at: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.
[38] Z. Yan and D.V. Sarwate, New systolic architectures for inversion and division in GF(2^m), *IEEE Computer* **52**(11) (2003), 1514–1519.
[39] R.W. Yeung and Z. Zhang, Distributed source coding for satellite communications, *IEEE Information Theory* **45**(4) (1999), 1111–1120.
[40] Y. Zhang, D. Chen, Y. Choi, L. Chen and S.-B. Ko, A high performance ECC hardware implementation with instruction-level parallelism over GF(2^{163}), *Microprocessor and Microsystems* **34**(6) (2010), 228–236.

Sunwoo Kim received the B.S. and M.S. degrees in the School of Electrical and Electronic Engineering from Yonsei University in 2009 and 2011, respectively. He was a research assistant in the Embedded Systems and Computer Architecture Laboratory. He is currently working in Electronics Platform Development Team, Hyundai Motor Company. His research interests include parallelization of network coding, parallel algorithm development, and high-performance hardware design.

Won Woo Ro received his B.S. degree in Electrical Engineering from Yonsei University, Seoul, Korea, in 1996. He received the M.S. and Ph.D. degrees in Electrical Engineering from the University of Southern California in 1999 and 2004, respectively. He worked as a research scientist in Electrical Engineering and Computer Science Department in University of California, Irvine. He currently works as an Assistant Professor in the School of Electrical and Electronic Engineering at Yonsei University. Prior to joining Yonsei University, he had worked as an Assistant Professor in the Department of Electrical and Computer Engineering of the California State University, Northridge. His industry experience also includes a college internship at Apple Computer Inc. and a contract software engineer in ARM Inc. His current research interests are high-performance microprocessor design, compiler optimization, and embedded system designs.