

means of simulation. The result is as shown in Fig .1.

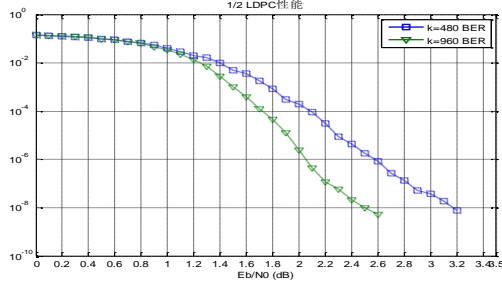


Figure 1. BER performance of LDPC coding for different frame lengths

Based on an analysis and comparison of the simulation results, it was found that 1/2 LDPC scheme required the SNR to be 3.2dB when the message length was 480 bits and the BER was 1E-8, which achieved 8.8dB coding gain in comparison with the non-coding scheme. It required the SNR to be 2.6dB when the message length was 960 bits and the BER was 1E-8, which achieved 9.4dB coding gain in comparison with the non-coding scheme. In other words, the 960 bits message length improve the coding gain by 0.6dB comparing with the 480 bits message length.

III. LDPC ENCODING ALGORITHM AND IMPLEMENTATION

A. Two-way Recursion Fast Coding Algorithm Selecting a Template^[7,8]

The QC-LDPC codes described above adopt dual-diagonal parity structure, which makes it possible to complete fast coding in a way of cyclic recursion. A code \mathbf{c} is divided into input information sequence \mathbf{s} and check sequence \mathbf{p} that is generated in coding process. \mathbf{s} and \mathbf{p} are further decomposed into k_b and m_b groups of z -dimension sub-vectors. Therefore, a code may be expressed as $\mathbf{c} = [\mathbf{s} | \mathbf{p}] = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{L}, \mathbf{s}_{k_b-1}, \mathbf{p}_0, \mathbf{p}_1, \mathbf{L}, \mathbf{p}_{m_b-1}]$, where $\mathbf{L}_{i,j}$ represents the sub-matrix corresponding to the element defined by Row i and Column j in matrix \mathbf{H}_{b_i} . Let's assume $h_b(0) = h_b(m_b-1) = l$, $h_b(x) = l_x$ and

$$\mathbf{b}_i = \sum_{j=0}^{k_b-1} \mathbf{L}_{i,j} \mathbf{s}_j \quad (0 \leq i \leq m_b-1) \quad (1)$$

Then, due to $\mathbf{H} \times \mathbf{c}^T = \mathbf{0}$, it is possible to deduce that

$$\mathbf{b}_0 + \mathbf{p}_0^{(l)} + \mathbf{p}_1 = \mathbf{0} \quad (2)$$

$$\mathbf{b}_1 + \mathbf{p}_1 + \mathbf{p}_{i+1} = \mathbf{0} \quad (1 \leq i \leq m_b-2, i \neq x) \quad (3)$$

$$\mathbf{b}_x + \mathbf{p}_0^{(l_x)} + \mathbf{p}_x + \mathbf{p}_{x+1} = \mathbf{0} \quad (4)$$

$$\mathbf{b}_{m_b-1} + \mathbf{p}_0^{(l)} + \mathbf{p}_{m_b-1} = \mathbf{0} \quad (5)$$

Where, $\mathbf{p}_i^{(l)}$ represents the vector of \mathbf{p}_i after shifting l times rightwards circularly.

If the m_b equations (2) through (5) corresponding to different rows are added together, we will obtain:

$$\mathbf{p}_0^{(l_x)} = \sum_{i=0}^{m_b-1} \mathbf{b}_i \quad (6)$$

After \mathbf{p}_0 is worked out, it is possible to figure out the following equation by means of forward recursion on the basis of Equation (2)

$$\mathbf{p}_1 = \mathbf{b}_0 + \mathbf{p}_0^{(l)} \quad (7)$$

In the meantime, it is possible to figure out the following equation by means of backward recursion on the basis of Equation (5).

$$\mathbf{p}_{m_b-1} = \mathbf{b}_{m_b-1} + \mathbf{p}_0^{(l)} \quad (8)$$

After $\mathbf{p}_1, \mathbf{p}_{m_b-1}$ is figured out, go on with the forward recursion to work out:

$$\mathbf{p}_2 = \mathbf{b}_1 + \mathbf{p}_1 \quad (9)$$

In the meantime, proceed with backward recursion to work out:

$$\mathbf{p}_{m_b-2} = \mathbf{b}_{m_b-2} + \mathbf{p}_{m_b-1} \quad (10)$$

Accordingly, it is possible to work out the intermediate check vector at last by means of two-way recursion.

$$\mathbf{p}_{m_b/2} = \mathbf{b}_{m_b/2} + \mathbf{p}_{(m_b/2)+1} \quad (11)$$

The sequential order of check vectors that are worked out with this two-way recursive algorithm is (\mathbf{p}_0) , $(\mathbf{p}_1, \mathbf{p}_{m_b-1})$, \mathbf{L} , $(\mathbf{p}_{(m_b/2)-1}, \mathbf{p}_{(m_b/2)+1})$ and $(\mathbf{p}_{m_b/2})$, where the parentheses mean the two check vectors may be solved simultaneously. Finally, it is possible to acquire the code in the form required by the system by combining the check vectors and the information vectors that are worked out together. In terms of hardware implementation, this two-way recursive coding algorithm is characterized by low computation complexity, high degree of parallelism, less resource consumption and low cabling complexity.

B. Pipelining Implementation of Quick Two-way Recursion Pipeline Algorithm^[9,10]

After all the \mathbf{b}_i vectors are figured out, work out the check vector \mathbf{p}_0 using Equation (6). And then, work out all the other check vectors $(\mathbf{p}_1, \mathbf{p}_{m_b-1})$, \mathbf{L} , $(\mathbf{p}_{(m_b/2)-1}, \mathbf{p}_{(m_b/2)+1})$ and $(\mathbf{p}_{m_b/2})$ one by one with the two-way recursive algorithm described in Section 1. Based on an analysis of the computational process of two-way recursion encoding algorithm, we developed a quick pipeline algorithm for the computation of $\mathbf{p}_1, \mathbf{L}, \mathbf{p}_{m_b-1}$. It can be seen from equations (7) through (11) that \mathbf{b}_i variables computation of different \mathbf{p}_i vectors with two-way recursive algorithm. $\mathbf{b}_0, \mathbf{b}_{m_b-1}$ is required for computation of $\mathbf{p}_1, \mathbf{p}_{m_b-1}$; $\mathbf{p}_2, \mathbf{p}_{m_b-2}$ are required for computation of $\mathbf{b}_1, \mathbf{b}_{m_b-2}$, and so on. Therefore, there is no access conflict for \mathbf{b}_i data in the parallel computation of \mathbf{p}_i . However, in the computation process of \mathbf{p}_i , the nesting dependence on its own makes it impossible for \mathbf{p}_i to complete synchronous computation.

Besides, the computation of $\mathbf{p}_1, \mathbf{p}_{m_b-1}$ depends on \mathbf{p}_0 and the computation of $\mathbf{p}_2, \mathbf{p}_{m_b-2}$ depends on $\mathbf{p}_1, \mathbf{p}_{m_b-1}$. Such nesting dependence forces the computation of \mathbf{p}_i to start from the both ends of the dual diagonals towards the middle in series. However, upon a further analysis, it is found that this dependency relationship is virtually nesting dependence between components of different \mathbf{p}_i vectors rather than between \mathbf{p}_i vectors. After the first components of $\mathbf{p}_1, \mathbf{p}_{m_b-1}$ are figured out, it is possible to compute the first components of $\mathbf{p}_2, \mathbf{p}_{m_b-2}$ and not necessary to wait the completion of the entire $\mathbf{p}_1, \mathbf{p}_{m_b-1}$ vector computation. Therefore, it is feasible to carry out approximative synchronous computation of \mathbf{p}_i by inserting the pipeline algorithm. The pipeline work flow for computation of check vector is as shown in Fig .2 when $m_b=8$. Fig .3 shows the clock sequence in which the check vectors were worked out. It needs $z+m_b-1$ clock cycles altogether to work out $\mathbf{p}_1, \mathbf{p}_{m_b-1}$ with this quick pipeline algorithm, which requires only m_b-1 more clock cycles than z that is needed for purely synchronous computation of \mathbf{p}_i . Just for comparison, it needs a total of $z \times m_b$ clock cycles if the pipeline algorithm is not used. Another benefit brought along with this pipeline architecture is the synchronous output of \mathbf{p}_i , i.e., instant output at the end of \mathbf{p}_i check-bit computation. It does not require buffer memory for vector \mathbf{p}_i , consequently to save storage resource. The encoder presented here in this paper adopts such an output design.

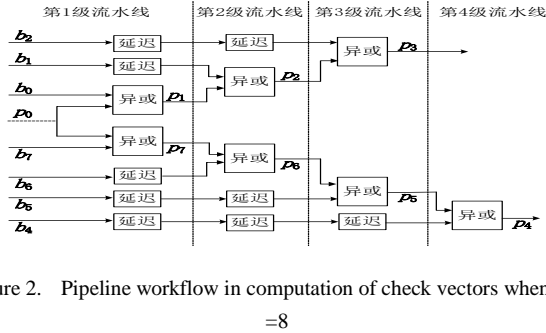


Figure 2. Pipeline workflow in computation of check vectors when $m_b=8$

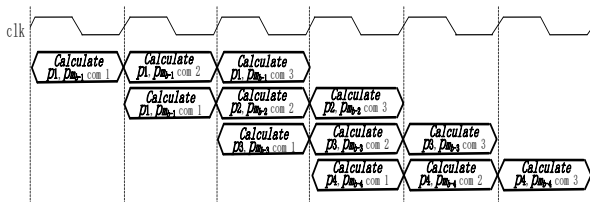


Figure 3. Computation timing diagram of parity-check vectors

IV. PREPARE YOUR PAPER BEFORE STYLING

A. BP Algorithm of Logarithm Domain(LLR-BP)

Decoding algorithm is purposed to acquire correct codes from the received messages as well as the relationship between nodes and parity-check nodes. For the convenience of description, the following notations are defined in the first place.

V_j =Variable node linked to check node f_j

V_j/i =Variable node linked to check node $f_j \setminus$ Variable node c_i

$=$ Check node linked to variable node c_i

$=$ Check node linked to variable node $c_i \setminus$ Check node f_j

$Mv(\sim i)$ =Messages coming from all the variable nodes except for c_i

$=$ Messages coming from all the check nodes except for f_j

$p_i = \Pr(c_i = 1 | y_i)$

S_i =All the parity-check equations containing c_i satisfy this event.

$q_{ij} = \Pr(c_i = b | S_i, y_i, Mc(\sim j))$, $b \in \{0,1\}$ (12)

$r_{ij}(b) = \Pr(\text{Parity-check equation } f_i \text{ satisfies } c_i = b, Mv(\sim i))$, $b \in \{0,1\}$.

For additive white Gaussian noise channel, let's set $x_i = 1 - 2c_i$ to serve as the binary value of the i th transmission. Thus, $x_i = +1(-1)$ when $c_i = 0(1)$. After that, replace c_i with x_i . Now let's define the log likelihood ratio.

$$L(c_i) = \log \left(\frac{\Pr(c_i = 0 | y_i)}{\Pr(c_i = 1 | y_i)} \right) \quad (13)$$

$$L(r_{ji}) = \log \left(\frac{r_{ji}(0)}{r_{ji}(1)} \right) \quad (14)$$

$$L(q_{ij}) = \log \left(\frac{q_{ij}(0)}{q_{ij}(1)} \right) \quad (15)$$

$$L(Q_i) = \log \left(\frac{Q_i(0)}{Q_i(1)} \right) \quad (16)$$

Thereby, it is possible to know the main steps of the logarithm-domain BP decoding algorithm(LLR-BP) as follows:

1) Initialization

For every received variable node, work out the initial channel information with the following equation.

$$L(q_{ij}) = L(c_i) = 2y_i / \sigma^2 \quad (17)$$

In the meantime, initialize the iteration times.

2) Updating check node

For all the check nodes f_j and their adjacent variable nodes $c_i \in V_j$, work out the messages $\{L(r_{ij})\}$ transmitted to the check nodes from the variable nodes.

$$\begin{aligned} L(r_{ji}) &= \prod_{i' \in V_j \setminus i} \alpha_{i'j} \cdot 2 \tanh^{-1} \left[\prod_{i' \in V_j \setminus i} \tanh\left(\frac{1}{2} \beta_{i'j}\right) \right] \\ &= \prod_{i' \in V_j \setminus i} \alpha_{i'j} \cdot 2 \tanh^{-1} \log^{-1} \log \left[\prod_{i' \in V_j \setminus i} \tanh\left(\frac{1}{2} \beta_{i'j}\right) \right] \\ &= \prod_{i' \in V_j \setminus i} \alpha_{i'j} \cdot 2 \tanh^{-1} \log^{-1} \sum_{i' \in V_j \setminus i} \log \left[\tanh\left(\frac{1}{2} \beta_{i'j}\right) \right] \end{aligned} \quad (18)$$

Where, $\alpha_{i'j} = \text{sign}(q_{i'j})$ and $\beta_{i'j} = |q_{i'j}|$.

3) Updating variable nodes

$$L(q_{ij}) = L(c_i) + \sum_{j \in C_i \setminus j} L(r_{ji}) \quad (19)$$

4) LQ updating

On completion of all the non-zero node computation, work out LQ for all the row number i .

$$L(Q_{ij}) = L(c_i) + \sum_{j \in C_i} L(r_{ji}) \quad (20)$$

5) Decision processing. Work out \hat{c}_i for each row:

$$\hat{c}_i = \begin{cases} 1 & L(Q_i) < 0 \\ 0 & \text{other} \end{cases} \quad (21)$$

Stop iterating if $\hat{c}H^T = 0$ or the iteration reaches the maximum number; otherwise, go back to Step 2 to start next iteration.

B. Modification of LLR-BP Algorithm

Since \tanh and its inverse operation are involved in the updating equation of check node in the LLR-BP algorithm, hardware implementation is not easy to achieve but with look-up table method.

MinSum algorithm is a decoding algorithm with regard to the excessive complexity of $\varphi(x)$ that is used in the check node updating in LLR-BP decoding algorithm. Considering the information updating characteristic of check nodes and variable nodes in the probability-domain and logarithm-domain BP decoding algorithm, MinSum algorithm replaces the log computation, multiplication and division with addition, subtraction and comparison. These changes lower the decoding complexity significantly while the decoding performance does not run down much in comparison with the LLR-BP decoding algorithm.

let's define $\varphi(x) = -\log[\tanh(x/2)] = \log(e^x + e^{-x} - 1)$. According to the function character, it is known that $\varphi^{-1}(x) = \varphi(x)$ and $x > 0$. And then,

$$L(r_{ji}) = \prod_{i \in V_j \setminus i} \alpha_{i,j} \varphi \left[\sum_{i \in V_j \setminus i} \varphi(\beta_{i,j}) \right] \quad (22)$$

Because it is hard to achieve hardware implementation for function $\varphi(x)$, it is required to initialize the function by means of approximation for practical application. Function $\varphi(x)$ has one characteristic as follows: the function value falls pretty fast along with the growth of x . Therefore,

function $\varphi \left[\sum_{i \in V_j \setminus i} \varphi(\beta_{i,j}) \right]$ almost depends on the minimum of

$\beta_{i,j}$ completely, that is, $\sum_{i \in V_j \setminus i} \varphi(\beta_{i,j}) \approx \varphi(\min_{i \in V_j \setminus i} \beta_{i,j})$. For this

reason, the updating of check node may be simplified as:

$$L(r_{ji}) = \left(\prod_{i \in V_j \setminus i} \text{sign}(Lq_{ij}) \right) \min_{i \in V_j \setminus i} (\beta_{i,j}) \quad (23)$$

After processing by this step, the hardware implementation will be facilitated, for the computational workload and the memory capacity is saved a lot. This processing step is also the main difference between MinSum algorithm and LLR-BP algorithm. However, this characteristic also brings about performance loss to some extent. The performance loss is especially considerable at low code rate. To enhance the performance of MinSum algorithm, several modified MinSum algorithms are proposed as follows.

Modified Algorithm 1: Normalized-MS(Normalized-MinSum algorithm)

At the updating step of check nodes, the updating equation becomes:

$$\Lambda_{mn}^{(k)} = \alpha \cdot \prod_{n \in N(m) \setminus n} \text{sign}(\lambda_{mn}^{(k-1)}) \cdot \min_{n \in N(m) \setminus n} |\lambda_{mn}^{(k-1)}| \quad (24)$$

$$L(r_{ji}) = \alpha \cdot \prod_{i \in V_j \setminus i} \text{sign}(Lq_{ij}) \min_{i \in V_j \setminus i} (\beta_{i,j}) \quad (25)$$

Where, α is usually a constant less than 1, of which the specific value must be analyzed with density evolution analytic technique based on different SNRs and iteration times. This modified algorithm is purposed to minify the updating value of check nodes with α so as to approach the updating value of check nodes in LLR-BP algorithm.

Modified Algorithm 2: Offset-MS(Offset-based MinSum Algorithm)

At the updating step of check nodes, the updating equation becomes:

$$\Lambda_{mn}^{(k)} = \prod_{n \in N(m) \setminus n} \text{sign}(\lambda_{mn}^{(k-1)}) \cdot \max \left\{ \min_{n \in N(m) \setminus n} |\lambda_{mn}^{(k-1)}| - \beta, 0 \right\} \quad (26)$$

$$L(r_{ji}) = \prod_{i \in V_j \setminus i} \text{sign}(Lq_{ij}) \cdot \max \{ \min_{i \in V_j \setminus i} (\beta_{i,j}) - \eta, 0 \} \quad (27)$$

Where, η is usually a positive constant, which must be optimized with density evolution analytic technique based on different SNRs and iteration times. This modified algorithm is purposed to determine if the updating value of check nodes in MinSum algorithm is useful or not by means of comparing both $\min_{n \in N(m) \setminus n} |\lambda_{mn}^{(k-1)}|$ and η , so as to approach the updating value of check nodes in LLR-BP algorithm.

In this modified algorithm, the check nodes will not be updated when $\min_{n \in N(m) \setminus n} |\lambda_{mn}^{(k-1)}| \leq \eta$; therefore, its algorithm performance is somewhat worse than NMS algorithm performance. However, this algorithm is easy to be implemented, for it does not require multiply operation.

V. PERORMANCE EVALUATION OF CODING AND DECODING MODULE

The resources occupied by the coding module are as shown below:

Resource	Value
File Status	Successful - Mon Mar 10 11:29:08 2016
Source File Version	0.1 Build 100 10/03/2016 EP FPGS Version
Device Name	lqps_code
Top-Level Entity Name	lqps_code
Family	Stratix II
Device	EP2S60F102014
Timing Mode	Fixed
Net timing requirements	Yes
Logic utilization	0.3%
Combinational ALUTs	1,505 / 40,352 (0.4%)
Dedicated logic registers	1,815 / 40,382 (0.4%)
Total read logic	3515
Total pins	0 / 718 (0.0%)
Total virtual pins	0
Total block memory bits	10 / 2,144,100 (0.0%)
EEP block memory bits	0 / 380 (0.0%)
Total FLLs	0 / 10 (0.0%)
Total DLLs	0 / 2 (0.0%)

Figure 4. The resources occupied by the coding module

Adaptive look-up table(ALUT) and memory are two important indicators in FPGA resource evaluation. The above figure showed that a total of 1505 ALUTs were occupied when EP2S60F102014 chip was used for coding implementation, which accounted for 3% of the total chip resources.

The resources occupied by the decoding module are as shown below when Normalized-MS algorithm is adopted.

File Status	Successful - Mon Mar 19 10:50:24 2012
Quartus II Version	9.1 Build 163 10/20/2009 EE Pack Version
Switch Name	10m01
Top-Level Entity Name	LDPC
Family	Stratix II
Device	EP2S60F1020I4
Timing Models	Final
Post-timing requirements	Yes
Logic utilization	2 %
Configured ALUTs	9,249 / 48,392 (19 %)
Configured Single-ported Registers	6,372 / 48,392 (13 %)
Total registers	6,372
Total pins	630 / 740 (85 %)
Total virtual pins	0
Total Block memory bits	59,040 / 2,944,192 (2 %)
8K Block memory elements	0 / 368 (0 %)
Total Fitter	1 / 12 (8 %)
Total Size	0 / 2 (0 %)

Figure 5. The resources occupied by the decoding module

ALUT and memory are two important indicators in FPGA resource evaluation. The above figure showed that a total of 9249 ALUTs were occupied when EP2S60F1020I4 chip was used for decoding implementation, which accounted for 19% of the total chip resources; a total of 59040 bit memory was occupied, which accounted for 2% of the total chip resources.

VI. PERFORMANCE EVALUATION OF CODING AND DECODING MODULE

In terms of coding delay, fast coding technique is used so that the coding process can be completed in about 20 clock cycles. The coding delay is approximately 2ms when the work clock of the coding module is 10KHz.

As for decoding delay, the decoding process needs about 544,450 clock cycles under the condition of 50 iterations and using the Normalized-MS algorithm. When the work clock of the decoding module is 40MHz, the decoding delay is 13.6ms; when the work clock is 60MHz, the decoding delay is 9.1ms.

VII. EVALUATION OF DECODING PERFORMANCE

A separate test was given to the LDPC decoding module and the test results were compared with MATLAB simulation curve to verify the module correctness and estimate the implementation cost of coding and decoding. The test block diagram is as shown in Fig. 6.

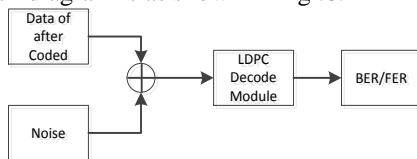


Figure 6. Block diagram of LDPC decoding performance test

The demonstration tests were given when the E_b/N_0 was 0, 0.5, 1, 1.5 and 2 respectively. In the testing process, MATLAB was used for generation after the E_b/N_0 was determined. A fixed frame data were inserted with noise after coding. On completion of 8bit quantization, the signal vector and the noise vector were stored in a ROM register. And then, read the data from two ROMs at 10KHz clock. Added them together before sending them into the LDPC decoding module. Acquired the BER and FER at the end

of decoding. Compared the test results with the simulation results generated by MATLAB to end up the validation test of LDPC decoding module. The test results are as shown in Fig. 7.

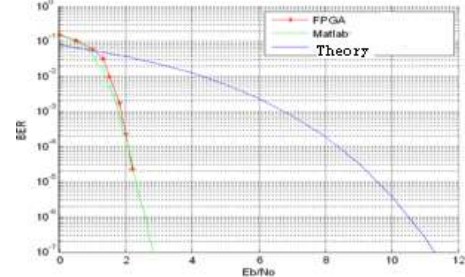


Figure 7. Performance test of LDPC decoding module

VIII. CONCLUSION

Conclusively, LDPC coding and decoding module is capable of satisfying the design requirements of navigation signal in terms of resource expenditure, delay constraint and coding gain.

REFERENCES

- [1] R.G.WEI, H.CHEN, J.HQIU, Z.S.HAO, G.X.LEI, Research on LDPC Codes Decoding Algorithm for High Data Rate Transmission, Radio Engineering, vol.41, No.3, Mar.2011, pp20—22.
- [2] D.J.C. MacKay and R.M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," Electron. Lett., vol.32, Aug.1996, pp1645-1646, DOI:10.1049/el:19961141
- [3] L.Z. W, L.CHEN, L.Q.ZENG, et al, "Efficient Encoding of Quasi-Cyclic Low-Density Parity-Check Codes," IEEE Transactions on Communications, vol.54, No.1, 2006, pp71—81.
- [4] R.G.GALLAGE, "Low Density Parity Check Codes," IRE Transactions on Information Theory, vol.8, No.8, 1962, pp208—220.
- [5] Y.KOU, S.LIN, M.P.C. FOSSORIER, "Low-density Parity-check Codes Based on Finite Geometries: a Rediscovery and New Results," IEEE Transactions on Information Theory, vol.47, No.7, 2001, pp2711—2736.
- [6] T.J. Richardson, "Efficient Encoding of Low-density Parity-check Codes," IEEE Trans on Information Theory, vol.47, No.2, 2001, pp638-656, doi:10.1109/18.910579.
- [7] R.J. SUN, "Research and application of LDPC decoding method," dissertation of Nanjing University of Science and Technology, 2012.
- [8] J.P. YANG, Q.C. CHEN, "Design and Implementation of 802.16e Standard LDPC Decoder," COMMUNICATIONS TECHNOLOGY, vol.43, No.5, 2010, pp18-19.
- [9] S.B. HE, C.H. LIU, Y.M. LIN, "Application of LDPC Codes in Satellite Navigation Systems", SPACECRAFT ENGINEERING, vol.18, No.3, 2009.05, pp72-76.
- [10] H. QIAO, W. GUAN, M.K. DONG, H.G. XIANG, "Design and Implementation of LDPC Decoder with High Throughput", Acta Scientiarum Naturalium Universitatis Pekinensis, vol.2, No.2, 2007, pp1-6.