ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.1, Pages : 346 – 352 (2013)
Special Issue of ICACSE 2013 - Held on 7-8 January, 2013 in Lords Institute of Engineering and Technology, Hyderabad

Reduction of Latency in an Asynchronous Communication System with Error Correction Capabilities

[1]Ms. Kasam Navya, [2]Dr. Syed Amjad Ali

[1]M.Tech. (ES & VLSID) student, Lords Institute of Engineering and Technology, Hyderabad, India. Email: kasamnavya@gmail.com
[2]Professor, Department of ECE and Dean Academics, Lords Institute of Engineering and Technology, Hyderabad India
Email: syedamjadali.lords@gmail.com.

ABSTRACT

This paper introduces a new family of error – correction unordered (ECU) codes for global communication, called zero – sum. It is the combination of delay insensitive codes and fault tolerance of error correcting codes. Two important feature of this code are that they are systematic and weighted. For this code, a wide variety of weight assignments is used. Two enhancements are also proposed for zero-sum code. The zero-sum^{+} code, it gives error detection for 3-bit errors, or alternatively provides 2-bit detection and 1-bit correction. The zero-sum* code, provides heuristic 2-bit correction while still guaranteeing 2-bit detection, under different weight assignments. The outline for this block level system micro architecture is given. In comparison with other ECU codes, the zero-sum code provides better or comparable coding efficiency, with 5.74% - 18.18% reduction in average number of wire transitions. Zero-sum* code provides 2-bit correction coverage, with 52.92% to 71.16% of all 2-bit errors with only a moderate decrease in number of bits per wire and increase in wire transitions.

Key words : Asynchronous communication, delay - insensitive, error-correcting codes, fault tolerances, unordered.

1. Introduction:

As digital systems grow in complexity, the challenges of design reuse, scalability, power and reliability continue to grow at a rapid pace [16]. They are expected to become major bottlenecks in less than a decade.

To provide flexibility in system integration, asynchronous global communication [4] [12] [13] is used such system forms a globally – asynchronous locally - synchronous (GALS) [8] [17] system. Applications of asynchronous global communication, using delay insensitive codes [18], include high-speed commercial FPGA's and Ethernet routing chips.

The contributions of this paper are, use of error correcting unordered codes in asynchronous communication, called zero-sum [1], [2]. These code supports the delay insensitivity (i.e., unordered property) and fault tolerance (i.e., 1-bit correction and 2-bit detection). This detection and correction of errors are based on their minimum hamming distances.

The zero-sum codes are systematic [6], [11] i.e., allows direct extraction of data by the receiver without any hardware. They are also weighted i.e., check field is represented as sum of data field index weights.

The outline of system micro architecture includes three key hardware support blocks which are an encoder (for the sender), completion detector (CD) and error-correcting unit (for the receiver).

The basic method for generating the zero-sum code builds on a prior approach briefly introduced by Berger [5] and Bose [6]. These approaches are limited for single index weight assignment. For these codes neither hardware support not experimental evaluation was performed.

2. Background and Related Work:

A. Point-To-Point Asynchronous Communication:

The proposed method assume point-to-point communication [4], [12], [13] & [18] between a sender and receiver.

a) Asynchronous Communication Channels:

Asynchronous communication means only flow of data between two entities and no clock information is present in this.

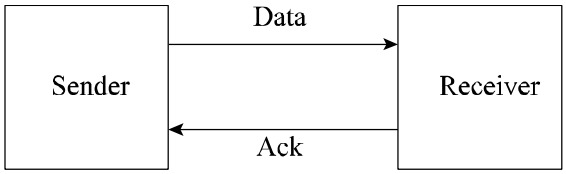

Figure 1: Point-to-point Asynchronous Communication

Fig. 1.shows the point-to-point communication. In this sender sends information to the receiver and the receiver in turn provide the Ack (Acknowledgement) to the ender. If the sender passes the actual data to the receiver, that data is typically replaced by the encoded data. The Ack indicates that the data has been received by the receiver and new data can be sent [18].

b) Four-phase Communication Protocol:

Given an asynchronous communication channel, a protocol is needed to transfer information from sender to the receiver. Most widely used protocol is four-phase or return-to-zero (RZ) [3], [16], [18].

Fig. 2 shows the four-phase or RZ protocol. This protocol has two phases. 1) Evaluate and 2) Reset. In evaluate phase logic 0 or logic 1 is evaluated. This protocol uses dual rail coding i.e., 1-bit of data represents two wires. When one signal is in data phase and other one is in reset phase then data should not be considered. When both signals are in data phase then data is to be considered. In this protocol we are

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.1, Pages : 346 – 352 (2013)
Special Issue of ICACSE 2013 - Held on 7-8 January, 2013 in Lords Institute of Engineering and Technology, Hyderabad

providing delay insensitivity with reset phase for every two data phases. This protocol has less complexity.

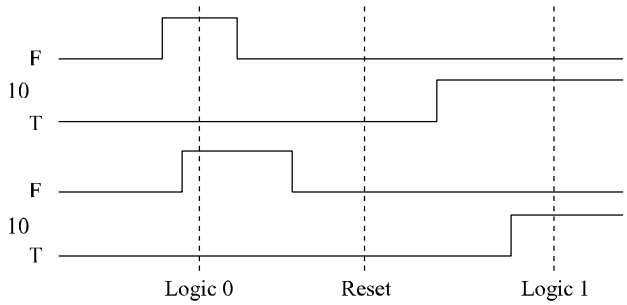

Figure 2: Four-phase (or) RZ Protocol

Data	T	F	
0	0	1	} data/evaluate phase
1	1	0	
	0	0	Reset phase

An alternative to the RZ protocol is two-phase or non-return to zero protocol [9], [12]. In this we don't have any reset phase. This protocol has even phase and odd phase. This also use the dual-rail coding. The codes designed for this typically have large overheads and it has more complex structure. Hence, four-phase communication protocol is used for communication in this paper.

c) DI Codes:

When asynchronous communication is used data must be suitably encoded so that the receiver can identify when a packet has been received. Delay insensitivity [3], [16], [18], means that the data comes to the receiver depends on the delays of individual bits in a code word. The key property is that no valid codeword is covered by another.

Definition 1 (unordered code [7]): A code word $X = x_1 x_2 \ldots x_n$ covers another code word $Y = y_1 y_2 \ldots y_n$ if and only if, for each bit position *i*, if $Y_i = 1$ then $X_i = 1$. In these case *Y* is covered by *X* or $Y \leq X$. A codeword `*c*' is unordered if each pair of code words in c is unordered.

There are two types of DI codes: Systematic and non-systematic codes. A systematic code [11], [18] contains separate data and check fields. For asynchronous communication, the check field provides extra bit to indicate the entire code is DI. Types of systematic codes are Berger [5] and Knuth [18] codes. In this no hardware decoders are required for the extraction of data because the original data directly appears in codeword.

In non systematic codes [4], [12], [18], there are no separate data and check fields. Data is encoded in a unified field, which ensures delay insensitivity. Dual-rail, 1-of- 4 and general class of *m*-of-*n* codes are example for these codes.

B. Hamming Error Correction Codes:

Hamming code is one of the mostly used error correcting code. It is the example of a weight-based code i.e., each bit position has a weight. These codes have data (information) and check (parity) bits. Data bits are assigned as non-power-of-two weights and check bits are assigned as power-of-two weights. The weights assigned for the data bits are called weight set. The parity bit provides error coverage for the data bits.

C. Basic Zero-Sum Code:

This section gives the detailed explanation for zero-sum code. This code uses the single index weight assignment. This code provides 1-bit correction or 2-bit detection for symmetric errors.

This code is combination of Hamming codes [10] and Berger codes [5]. In Berger code, '0' in data bit is used to generate the DI field. Berger code counts the number of 0's in data bit, where as zero-sum code adds the index weights of 0's in the data field.

a) Overview of Code:

Fig. 3 shows the examples of zero-sum code for 2-bit, 3-bit and 4-bit data words. The code has two fields. 1) data and 2) check bits. Each bit position is assigned one index weight. The data word is represented as non-power-of two weights and check bits are power-of-two weights. The sum of index weights of 0's in the data word is represented in check bits. The sum of index weights of check bits must be greater than the sum of index weight of data word.

b) Formal Calculation of Code Length:

The check field must support the binary representation of sum of data index weights i.e., to handle the extreme case where all bits in data word is 0's. Therefore, the total no. of check bits allocated is the $[\log_2 (\Sigma \text{ data word index weights})] + 1$.

The check field length k in terms of data length `*n*' is

$$k = [\log_2 ([n + \mu) (n + \mu + 1)|2] - 2 \mu + 1)]$$

Example 1: For a 4-bit data word 1010 in fig. 2, the sum of data word index weight of 0's is `9'. the corresponding check field is the binary representation of 9, which is 01001. In similar way all the data words are represented in check bits.

c) Detecting and correcting a 1-bit Error:

Hamming code provides a syndrome which is a vector of individual check bits and zero-sum provides a syndrome which is a single positive integer. Syndrome is calculated as the difference between the sum of index weights of 1's in check bits and sum of index weights of 0's in data word. If syndrome value is `0' there is no error and syndrome value is non-zero indicates an error. It is also used to correct the 1-bit error. The syndrome value indicates the index weight of the corrupted bit and by inverting that bit we can correct the error.

Example 2: If suppose there is an error in transmitting the 4-bit data word 0010, due to a flip in data bit with index 6 (i.e., erroneous data word 0110), transmitted with original error free check field (i.e., 10000). The newly calculated check field based on corrupted data word is 10 (i.e., 7 + 3). The original check field is 16. Therefore the syndrome is 16

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.1, Pages : 346 – 352 (2013)
Special Issue of ICACSE 2013 - Held on 7-8 January, 2013 in Lords Institute of Engineering and Technology, Hyderabad

– 10 = 6, which is non-zero and non-power-of-two, indicates that the error occur in the data bit with index weight `6'.

D. Formal Proof: Zero-Sum a ECU Code:

	Data word		Check bits			
Index ←	i_5	i_4	i_3	i_2	i_1	i_0
Weights ←	5	3	8	4	2	1
	0	0	1	0	0	0
	0	1	0	1	0	1
	1	0	0	0	1	1
	1	1	0	0	0	0

(a)

	Data word			Check bits			
Index ←	i_6	i_5	i_4	i_3	i_2	i_1	i_0
Weights ←	6	5	3	8	4	2	1
	0	0	0	1	1	1	0
	0	0	1	1	0	1	1
	0	1	0	1	0	0	1
	0	1	1	1	0	0	0
	1	0	0	0	1	1	0
	1	0	1	0	1	0	1
	1	1	0	0	0	1	1
	1	1	1	0	0	0	0

(b)

	Data word				Check bits				
Index ←	i8	i7	i_6	i_5	i_4	i_3	i_2	i_1	i_0
Weights ←	7	6	5	3	16	8	4	2	1
	0	0	0	0	1	0	1	0	1
	0	0	0	1	1	0	0	1	0
	0	0	1	0	1	0	0	0	0
	0	0	1	1	0	1	1	0	1
	0	1	0	0	0	1	1	1	1
	0	1	0	1	0	1	1	0	0
	0	1	1	0	0	1	0	1	0
	0	1	1	1	0	0	1	1	1
	1	0	0	0	0	1	1	1	0
	1	0	0	1	0	1	0	1	1
	1	0	1	0	0	1	0	0	1
	1	0	1	1	0	0	1	1	0
	1	1	0	0	0	1	0	0	0
	1	1	0	1	0	0	1	0	1
	1	1	1	0	0	0	0	1	1
	1	1	1	1	0	0	0	0	0

(c)

Figure 3: Example of zero-sum ECU codes (a) 2-bit ECU b) 3-bit ECU (c) 4 –bit ECU.

Proof: By definition1, it is sufficient to show that zero-sum code, c, is unordered if each pair of code words is unordered. Given two data words *X* and *Y*, by definition 1 if *X* covers *Y*, then *X* has more 1's and *Y*, and *Y* has more 0's than *X*. The check field is computed as the sum of the data field index weight which are `0'. The sum of index weights of check field must be greater than the sum of index weights of data word.

Theorem 2 (Zero-sum code 1-bit correction [6]): Every zero-sum code provides 1-bit detection and correction.

Proof: The check bits can be represented as the sum of the index weight of 0's of data word. We calculate the syndrome value to verify that the data word and check bits are correct or not. If the syndrome is non-zero there is an error occurred. The value of the syndrome is the difference between the check fields of original codeword to the check field of the newly computed codeword.

Theorem 3 (Zero-sum code error detection [5]): Every zero-sum code provides 2-bit detection.

Theorem 1 (zero-sum code delay – insensitivity [5]): Every zero sum code is unordered.

Proof: The proof is immediate from theorem 2. Since all 1-bit errors can be corrected, the code must have minimum distance of at least `3'; therefore, all 2-bit errors can be detected.

570

3. Extending the Zero-Sum Code:

A zero-sum+ code: zero-sum+ code provide two alternative modes. In one mode, it can detect all odd number and up to 3-bit errors. In second mode, 2-bit error can be detected and 1-bit error can be corrected.

Overview: It is same as zero-sum code with extra parity bit. Both data word and check field are identical to those of a zero-sum code. The new bit is the parity bit which indicates the even parity.

No. of Errors	Parity	Syndrome Value
0	Even	Zero
1	Odd	Zero or non-zero
2	Even	Non-zero
3 (or odd)	Odd	Zero or non-zero

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.1, Pages : 346 – 352 (2013)
Special Issue of ICACSE 2013 - Held on 7-8 January, 2013 in Lords Institute of Engineering and Technology, Hyderabad

Table 1: Zero-sum$^+$ Error detection classification

As shown in table 1, each error type is detected by it parity and syndrome values. When the syndrome is zero and parity is even then there are no errors. When 1-bit error occurs, there are two possibilities, each resulting in odd parity. If a 1-bit error occurs in data word or check bits, the syndrome is non-zero, and if 1-bit error occur in parity bit the syndrome is zero. When the parity is even and syndrome is non-zero indicates 2-bit errors.

Error Correction:

No. of Errors	Parity	Error Handling
0	Even	None
1	Odd	1-bit correction method
1	Odd	Toggle parity bit

Table 2: Zero-sum$^+$ error correction Classification

Table 2 shows zero-sum error correction classification. When `0' error occurs, parity is even and syndrome value is zero. Therefore no error handling method I applied. If no. of errors is `1' then there are two cases. If the parity is odd, syndrome is non-zero then 1-bit correction method is applied. It is identical to the zero-sum the parity bit the error can be corrected

4. Hardware Support:

Hardware Components: Fig. 4 shows the block-level system micro architecture the four phase communication channel is used to transfer the data between sender and receiver. The sender node generatesthe data word to the encoder which generates the check bits. The corresponding data word and check bits are given to the completion detector and error correction unit. The CD generates the Ack which again given back to the sender. The ECU unit generates the corrected data word and the check bits.

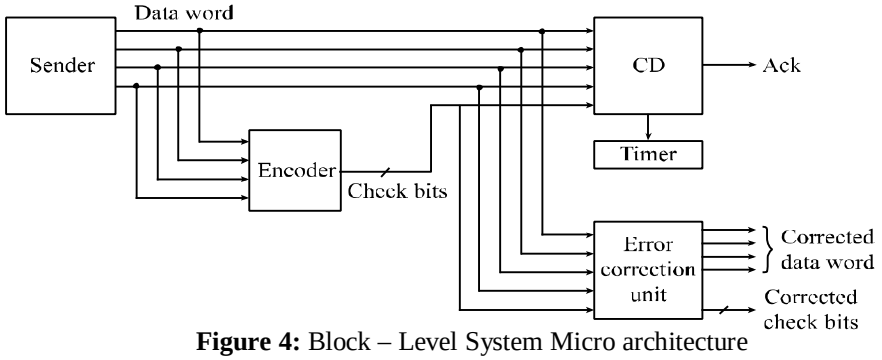

Figure 4: Block – Level System Micro architecture

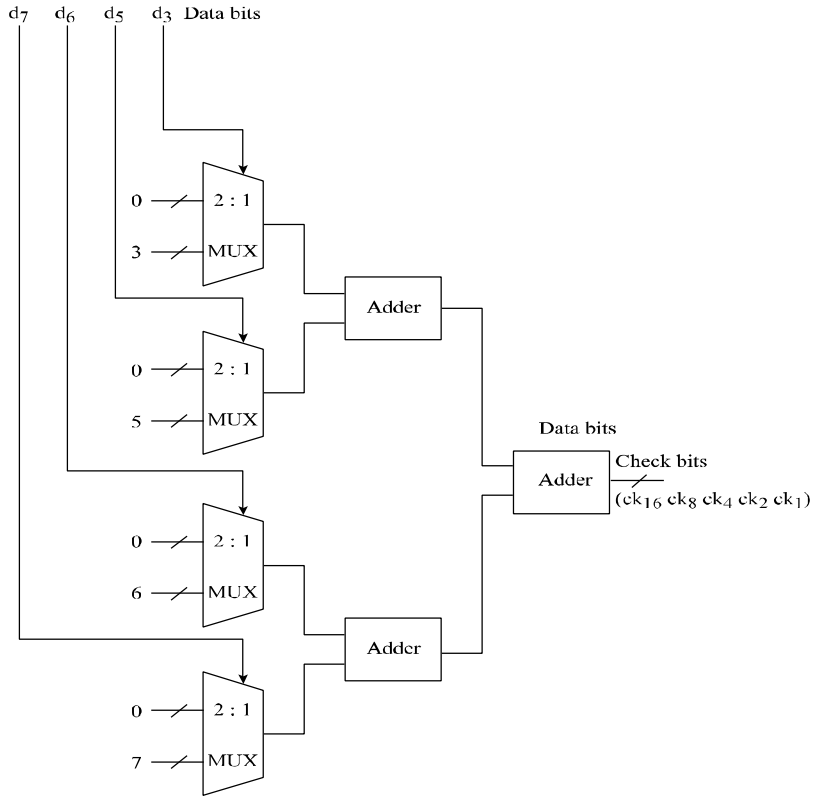

Figure 5: Basic 4-bit Encoder Design

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.1, Pages : 346 – 352 (2013)
Special Issue of ICACSE 2013 - Held on 7-8 January, 2013 in Lords Institute of Engineering and Technology, Hyderabad

Fig. 5. shows the basic 4-bit encoder designs. It consists of multiplexers followed by adders. There is one selector for each data field index value (3, 5, 6, 7). This selector passes the index value if the corresponding data bit is `0' and it passes `0' if the corresponding data bit is 1. By adding all these values the check bits can be generated.

In CD we have a timer block. This timer waits for the correct data to be sent. If the sender passes some information with one error then the CD waists for the arrival of correct bit and it waits for the valid code word. If the valid code word i sent then an Ack is given the sender. ECU unit generate the co-deword by detecting and correcting the 1-bit errors.

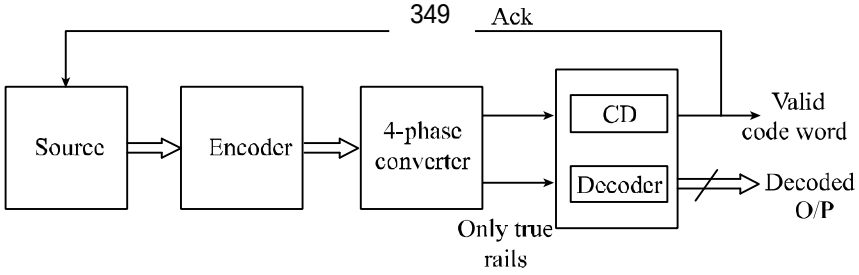

Figure 6: Modified Block Level System Architecture

Fig. 6 shows the modified block level system architecture. In this we have source, encoder, 4-phase converter and CD and decoder.

Initially we have Ack = 0, enable = 0. When enable is `1' source will give the information or data to the encoder. Encoder generates the check bits. The output of encoder is given to the 4-phase converter. In this 4-phase converter evaluate phase and reset phase are present. In this one bit can be represented as true rails and false rails. When both inputs are in data phase then only the data will be considered. If one is in evaluate phase and one is reset phase data will not considered. From this the total output is given the CD and only true rails are given to the decoder. In this CD, we are totally removing the timer block. The CD waits for the data word and once the data has received then it sends Ack to the sender. In this we are using a c-element whose output is `0' when all inputs are `0' and vice-versa. In the previous architecture, CD waits for the arrival of correct data but in this it waits only for the data and the decoder decodes the true rail and produces the output. By the removal of timer block we are reducing the execution time i.e., reducing the latency with 1-bit detection and correction for the zero-sum code.

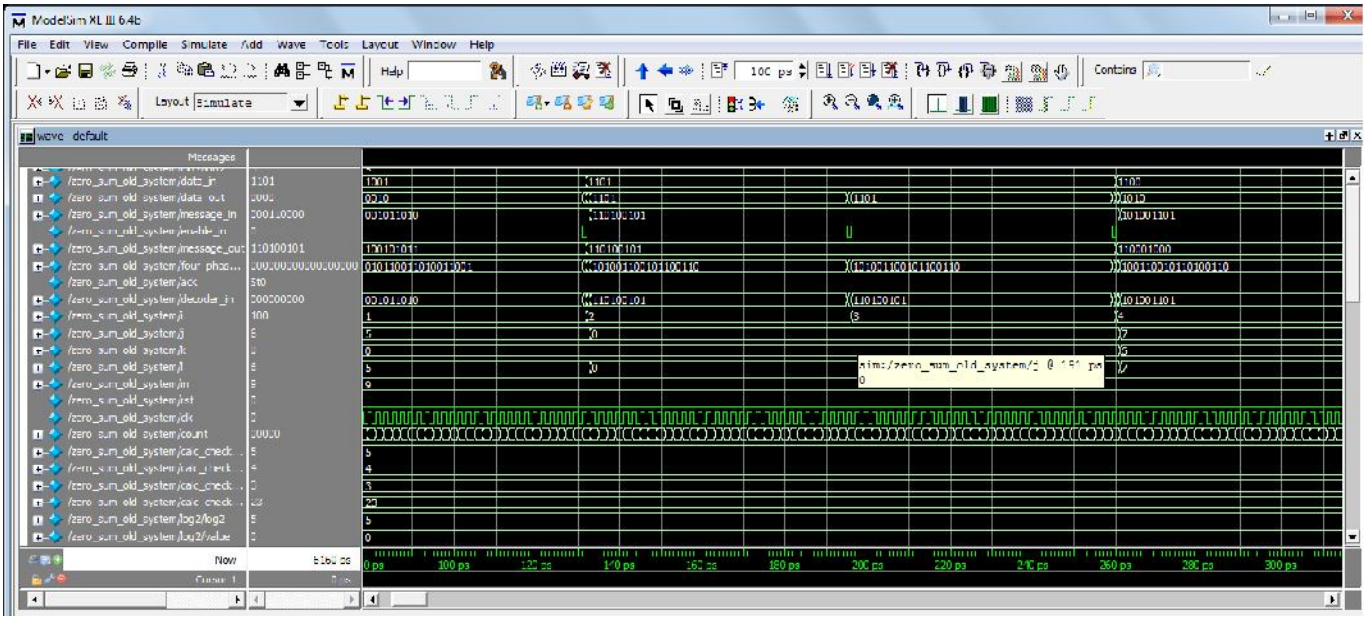

Figure 7: Zero sum system with timer block

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.1, Pages : 346 – 352 (2013)
Special Issue of ICACSE 2013 - Held on 7-8 January, 2013 in Lords Institute of Engineering and Technology, Hyderabad

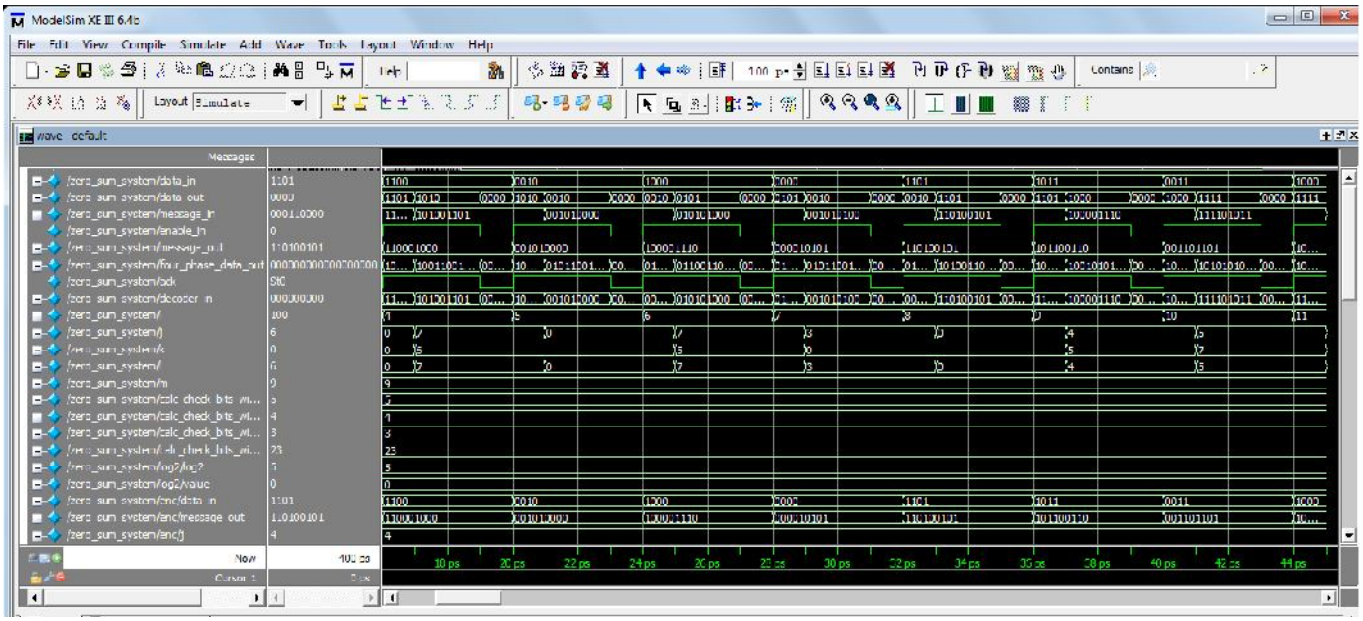

Figure 8: Zero sum system without timer block

6. Conclusion:

Zero-sum codes support the design of asynchronous global communication which combines the delay insensitivity and fault tolerance. The complete zero-sum code family was defined, by generated the weight assignment. Zero-sum+ and zero-sum* were also proposed.

Zero-sum+ code extends error detection to 3-bit errors and zero-sum* code provides 2-bit detection and correction coverage. In comparison with other ECU codes, zero-sum code provides better coding efficiency. The correction coverage of 2-bit errors are upto 52.92% to 71.16%. We identified the simulation results for zero-sum code and reduction of latency with a modification to the block level architecture.

REFERENCES

[1] M.Y. Agyekum, "Designing delay – insensitive codes for robust global asynchronous communication", Ph.D. dissertation, Columbia University, New York, Jan. 2011.

[2] M.Y. Agyekum and S.M. Nowick," An error-correcting unordered code and hardware support for robust asynchronous global communication", in proc DATE mar. 2010, PP, 765 – 770.

[3] V. Akella, N.H. Vaidya and G.R. Redinbo, " Asynchronous comparison based decoders for delay-insensitive code", IEEE trans. Comput., Vol. 47, No. 7, PP – 802 – 811, Jul, 1988.

[4] W.J. Bainbridge, W.B. Toms, D.A. Edwards and .B. Fuber, "Delay insensitive point-to-point interconnects using M-of-N codes", in Proc. IEEE async. Symp. May 2033, PP. 132 – 140.

[5] J.M. Berger " A note on error detecting codes for asymmetric channel", Inform, Contr. Vol. 4, No. 1, PP. 68 – 73, 1961.

[6] B. Bose, "unidirectional error-correction | detection for VLSI memory", in proc. ISCA, 1984, PP. 242 – 244.

[7] B. Bose and T.R.N. Rao, " Theory of unidirectional error correcting | detecting codes", IEEE trans, comput, vol. 31, No. 6. PP. 521 – 530, Jun 1982.

[8] D.M. Chapiro, "Globally – asynchronous locally synchronous system", Ph.D, thesis, Dept. Comput. Sci., Stanford Univ., Palo Alto, CA, Oct. 1984.

[9] M.E. Dean, T.E. Williams and D.L. Dill, " Efficient self-timing with level-encoded 2-phae dual rial (LEDR)", in proc. UC santa cruz conf. Adv. Res. VLSI, 19091, PP. 55-70.

[10] R.W. Hamming, "Error detecting and Error correcting codes", Bell syst. Tech. J., Vol. 29, No. 1, PP. 147 – 150, 1950.

[11] N.K. Jha,"Seperable codes for detecting unidirectional errors", IEEE trans. comput. Aided Des., Vol., 8, No. 5, PP. 571 – 574, May 1989.

[12] P.B. McGee, M. Y. Agyekum, M.A. Mohamed, and S.M. Nowick, " A level – encoded transition signaling protocol for high throughput asynchronous global communication", in proc. IEEE async. Symp. Apr. 2008, PP. 116 – 127.

[13] S.Ogg. B. Al-Hashimi, and A.Yakovlev, "Asynchronous transient resilient links for NOC, "in proc. CODEs, Oct. 2008, PP. 209-214.

[14] W.W. person and E.J. Weldon, Error-correcting codes, 2nd ed. Cambridge, MA; MIT press, 1972.

[15] S.J. Piestrak and T. Nanya, "Towards totally self-checking delay insensitive system", in proc FTCS, Jun. 1995, PP. 228 – 237.

[16] J.D. Owens, W.J. Dally, R. Ho., D.N. Jayasimha, S.W. Keckler and L.S. Peh, "Research challenge for on-chip

573

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.1, Pages : 346 – 352 (2013)
Special Issue of ICACSE 2013 - Held on 7-8 January, 2013 in Lords Institute of Engineering and Technology, Hyderabad

interconnection networks", IEEE micro., Vol. 27, No. 5, PP. 96 – 108.

[17] P. Techan, M. Greenstreetn, and G. Lemieux, "A survey and taxonomy of GALS design style", IEEE des. test comput., Vol. 24, No. 5, PP. 418 – 429. Sept-Oct 2007.

[18] T.Verhoeff, "Delay insensitive codes" An overview", Distrib comput., Vol 3, No. 1, pp. 1 – 8, 1998.