Cyclic Redundancy Check: A Novel Software Implementation for machine cycle optimization and analysis for deciding Low Peak to Average Power Ratio Discrete Sequences

A. Kotade[1], A. Nandgaonkar[1] , S. Nalbalwar[1] and U. Shiurkar[2]

[1]Electronics and Telecommunication Engineering Department, Dr. BATU, Lonere-402103, Raigad, Maharashtra, India.
[2]Director, Deogiri Institute of Engineering and Management Studies, Aurangabad – 431005
{kotade.amol@gmail.com; abnandgaonkar@yahoo.com; slnalbalwar@gmail.com}

Abstract: Cyclic Redundancy Check (CRC) is widely used error detection technique in many contemporary communication systems such as Fourth Generation (4G) Mobile Communication-Long Term Evolution (LTE) and LTE Advanced, Wi-Fi, Wireless LAN. For real time embedded systems, code size (Memory), Processor Machine Cycle (Speed) and Power are the three important parameters which are needs to be optimized. CRC is very effective and simple for error detection but its software implementation is not efficient. This paper presents software implementation of CRC using Bit by Bit (BYB) and Look-Up Table (LUT) approaches reported in the earlier literature. Using these approaches, we have compared machine cycle requirements for computation of CRC-3/5/8/12/16 generator polynomials. We have used TMS320C6713 and Freescale Star Core SC140 architectures for comparing the machine cycle requirements. Then we have intuitively modified our software implementations (Based on C program) of LUT using In Place Computation (IPC).This IPC-LUT based CRC computation is found to be more optimized in terms of machine cycle and memory compared to LUT method. We have reduced the machine cycle requirement by 39.47 % using our IPC-LUT approach compared to conventional LUT. We have also developed inline assembly code for SC140 architecture using IPC-LUT approach that takes only 45 machine cycles for computations. Peak to Average Power Ratio (PAPR) is one of the major drawback of contemporary communication systems. For third parameter (Power), we have simply done the analysis to fix up the decision criteria for deciding the sequences having low PAPR.

Keywords: ***Cyclic Redundancy Check (CRC), Look- Up Table (LUT), Machine Cycles, Peak to Average Power Ratio (PAPR), Optimization.***

1 Introduction

Emerging demands for high data rate services and high spectral efficiency are the key driving forces for the continued technology evolution in wireless communications. Third generation (3G) mobile communication systems have been commercially deployed to meet the initial demand for high data rate. Wireless communication for mobile terminals has been a high performance computing challenge. It requires almost super computer performance while consuming very little power [1]-[2]. This requirement is being made even more challenging with the move to Fourth and Fifth Generation (4G / 5G) wireless communication. Next generation data rates are greater than current 3G technology hence it will require more computational power. Leading technologies are protocols like 802.16e (Mobile Worldwide Interoperability for Microwave Access-WiMAX) and Third Generation Partnership Project (3GPP) Long Term Evolution (LTE), LTE [11] which uses Orthogonal Frequency Division Multiplexing (OFDM) at core level. A promising modulation technique that is increasingly being adopted in the telecommunication field is OFDM [3]. ODFM is a good solution for high speed digital communications. But high Peak to Average Power Ratio (PAPR) is a major problem in OFDM [8], [9], [10]. In OFDM, the data is spreaded over a large number of orthogonal carriers modulated at lower rates. The carriers can be made orthogonal by appropriately choosing the frequency spacing between them. Its advantages are high data rate and bandwidth efficiency. To provide high data rate in next generation wireless communication systems, the execution of all baseband processing algorithms must be done at high speed. The algorithms are implemented at Physical Layer. The physical layer deals with bit level transmission between different communicating stations. It consists of the basics networking hardware transmission technologies of a network.

B. Iyer, S. Nalbalwar and R. Pawade (Eds.)
ICCASP/ICMMD-2016. Advances in Intelligent Systems Research.
Vol. 137, Pp. 441-449.

Hence developing the algorithm which will take Minimum Machine Cycles for execution (High Speed), Minimum Code Size (Less Memory) and Minimum Power consumption is of prime consideration. Cyclic Redundancy Check (CRC) is widely used error detection method in data transmission and storage systems. It is simple but its software implementation is not efficient. Using CRC for error detection in embedded systems involves trade off among Speed (Machine Cycle), Memory (Code Size) and Power consumption. Because many embedded systems have significant resource constraints, it is important to understand the available trade off options and find the ways to attain better error detection at lower computational cost [6]-[7]. In this paper, we have studied the optimization of CRC computations in terms of machine cycle and memory requirement. CRC typically uses Galois Field, GF (2) for its operation. It is basically a discrete sequence. Hence we have also done PAPR analysis for discrete sequences for understanding the power constraints.

2 Physical Layer Context

The message carried over the physical channel is protected by various Forward Error Correcting Codes (FEC) in the physical layer. With FEC, redundant parity bits are added to the message, and these bits allow the receiver to detect and correct the errors. In the channel coding process CRC is appended to the input data packet and then passed to the FEC encoder. After encoding, puncturing is performed to increase the data rate followed by the interleaver to distribute the burst error. The scrambler introduces the pseudo random sequence into the incoming bit stream. This avoids the occurrence of long streams of zeros or ones and also provides better synchronization. At the receiver side exactly opposite operations are performed to get back the original information bits. The flow of data through different channel coding blocks can be referred from **Fig. 1**. The algorithm design and software implementation overview is given in the subsequent sections.

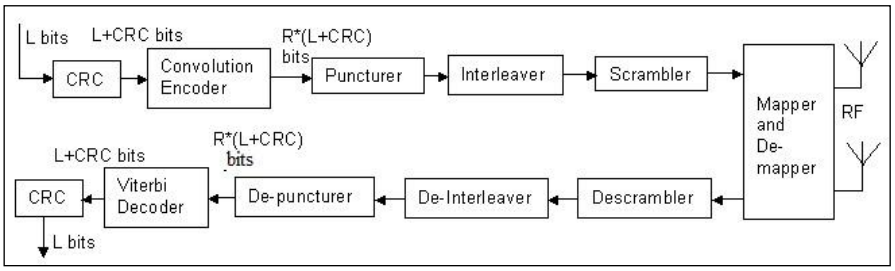


Fig. 1. Channel Coding Sub-blocks

2.1 A Novel Strategy for Algorithm Design

Fig. 1 shows all the sub-blocks of the channel coding block. Blocks are implemented in such a way that it exposes two Application Peripheral Interface (API). One of these is the *Initialization API* while second is the actual kernel of the block, or the *Processing API*. Typically, the user of the channel coding block would call the *Initialization APIs* for all the blocks at system start up thereby blocking the memory required by various blocks, initializing Look Up Tables (LUT) and other data structures. Thereafter, in the steady state operation, the user would call the *Processing API* as and when required. **Fig. 2** shows the call sequence for these APIs:

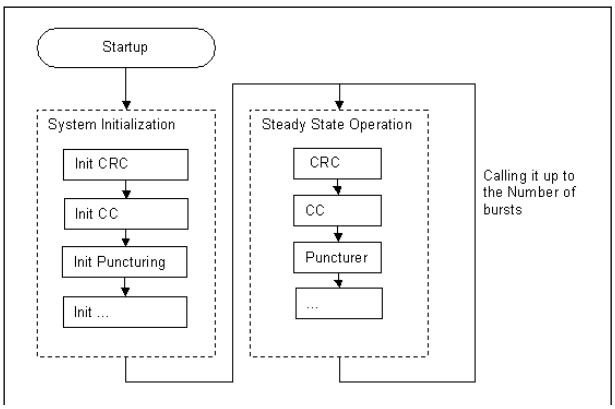


Fig. 2. Call Sequence of the Channel coding APIs

2.2 Algorithm Testing Framework

The test framework includes a set of test stub applications along with configurable parameters specific to the routines which are to be tested. Typically, a developed routine would be tested by building a project using Freescale CodeWarrior IDE. The input and reference output test vectors will be provided. The test stub application will call the *Initialization API* and then will pass the input test vector to the routine being tested (*Processing API*). It will then compare the output generated by the routines against the reference output test vector calculated using hand computations. It will provide the test results as SUCCESS or FAILURE depending on the final comparison. The test vectors (i.e. reference input and output) would be saved as ASCII text files with typically one value per line. The values can be unpacked bits (0 or 1), packed bytes/words (unsigned bytes/words) or soft values (signed bytes). The nature of values would depend on the input/output format of the routine to be tested. **Fig. 3** shows flow chart for test stub.

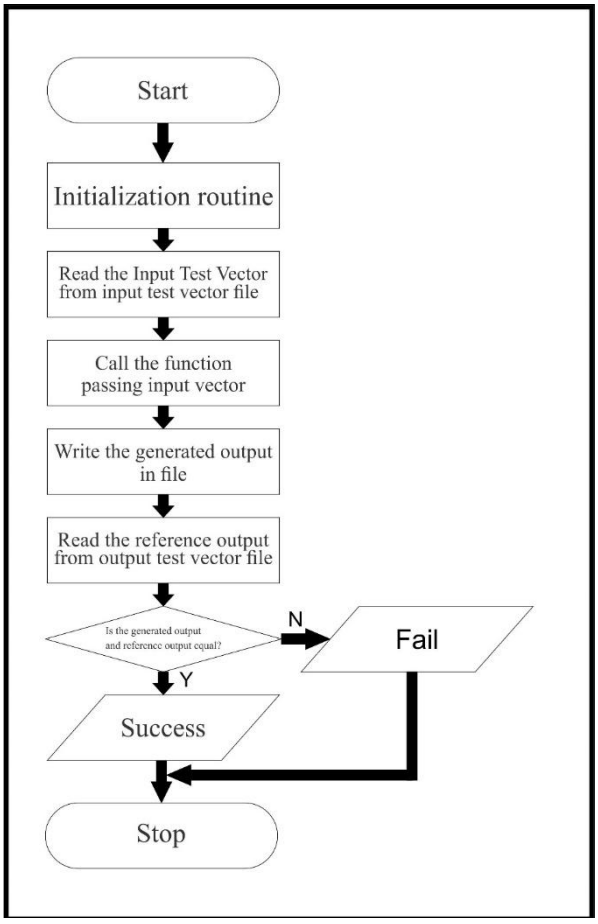


Fig. 3. Flow Chart for Test Stub

3 CRC

The CRC length that can be inserted has five different values: 3, 5, 8, 12 and 16 bits. Probability of undetected errors is low, when length of CRC is high.
Basic CRC computation algorithm: Bit by Bit Computation (BYB) [5]

1. Check MSB of data bit.
2. If MSB bit =1, then XOR the data with CRC poly and left shift by 1.
3. If MSB bit=0, then only shift data to left by 1.
4. After processing all bits, remainder is CRC.

Table 1. CRC polynomials

CRC	*Polynomial*
CRC-3	$1 + D + D^3$
CRC-5	$1 + D + D^2 + D^3 + D^5$
CRC-8	$1 + D + D^3 + D^4 + D^7 + D^8$
CRC-12	$1 + D + D^2 + D^3 + D^{11} + D^{12}$
CRC-16	$1 + D^5 + D^{12} + D^{16}$

3.1 Software Implementation

Optimized Data Structure

```
typedef struct
{
   UINT1 *u1InpPtr;   // Pointer to Input buffer
   UINT2 u2InLength;  // Length of input data in bits
   UINT2 u2CrcPolyLen;// Length of the CRC polynomial
   UINT2 u2CrcPolyEq; //CRC Polynomial (Drop MS bit)
   UINT2 u2CrcLut [256]; // Lookup Table for the CRC
} stCrc;
```

Output buffer pointer and output length parameters are not used in final optimized data structure. u2CrcLut parameter of structure stores the 256 entries corresponding to CRC for input byte varying from 0x00 to 0xFF. Length of each entry is equal to length of CRC polynomial. CrcLutGenerator ( ) function generates the LUT [5] specific to the CRC polynomial. As a part of initialization, members of the stCrc structure are initialized and the LUT is populated by calling the CrcLutGenerator () function. The structure parameter u1InPtr and u2InLength are initialized by user.

Pseudo Code:

```
INT2 CrcInit (stCrc *crc, UINT2 u2CrcPolyLen, UINT2 u2CrcPolyEq)
  {
     IF (crc=NULL) THEN
         RETURN (FAIL)
     crc->u2CrcPolyLen = u2CrcPolyLen; // Assign the
                            Poly length to structure
    //In the polynomial passed, LSB has D(0)and MSB has
      D(N),the function requires LSB to have D(N)and
      MSB to have D(0).Invert the polynomial for
      compatibility and initialize the structure with
      inverted polynomial (MSB becomes LSB)
     crc->u2CrcPolyEq = inverted u2CrcPolyEq

     CrcLutGenerator (crc) //Call the function to
                             generate the LUT
     RETURN (SUCCESS)
  }
```

The CRC encoder computes the CRC of the input message and appends it at the end of the input buffer. It processes one byte at a time and uses pre-computed values stored in the LUT.

CRC Encoder*:*

Pseudo Code:

```
INT2 CrcEnc (stCrc *crc)
{
```

```
u2Iteration = crc->u2InLength /8; //Determine the no. of
                                   i/p data bytes
u1NoOfBitsLeft = crc->u2InLength % 8; // No. of bits not
                                   falling in the byte
                                   boundary
   IF (u1NoOfBitsLeft) THEN
      u2Iteration++
   ENDIF
   u2Reg = 0            // Initialize the processing
                           register
   FOR i= 0 to u2Iteration DO
       Read next input data byte
       XOR the input data byte with lower order byte of
       u2Reg to get an INDEX
       Shift u2Reg 8 bits to the right
       XOR u2Reg with the contents of LUT at INDEX
       location
   ENDFOR
Append the CRC to input buffer with lower byte first followed by upper byte
RETURN (crc->u2InLength + crc->u2CrcPolyLen); //Returns
                     the total no. of bits in the output
}
```

CRC LUT Generator

It performs the bitwise operation to compute the CRC for all the input byte varying from 0x00 to 0xFF.

Pseudo Code:

```
static INT2 CrcLutGenerator (stCrc *crc)
{   IF (crc=NULL) THEN
      RETURN (FAIL)
   Compute the 16-bit mask as per the length of CRC
   polynomial
   FOR INDEX = 0 to 256 DO
      Load the u2Reg with INDEX value
      FOR i= 0 to 8 DO
        IF (u2Reg & 0x0001) THEN
            u2Reg = (u2Reg >> 1) ^ u2CrcPolyEq
        ELSE
            u2Reg = u2Reg >> 1
        ENDIF
      ENDFOR
      Mask the contents of u2Reg
      Copy the u2Reg in LUT at INDEX value
   ENDFOR
   RETURN (SUCCESS)
```

4 Results and Discussion

Table 2 provides the machine cycle requirements for the computation of different variants of CRC on Freescale SC140 architecture using conventional BYB, LUT [5]-[7] approaches and out proposed IPC-LUT approach. Optimization level of 0 and 3 can be set in SC140 Integrated Development Environment (IDE) project setting options. Finding the number of machine cycles, memory consumed, utilization of resources (shifters, Multiply and Accumulate-MAC etc) present in architecture for developed software program is termed as “Profiling”. BYB is traditional approach which consumes more machine cycles compared to LUT.

Table 2. CRC optimization results obtained using Freescale SC140 architecture

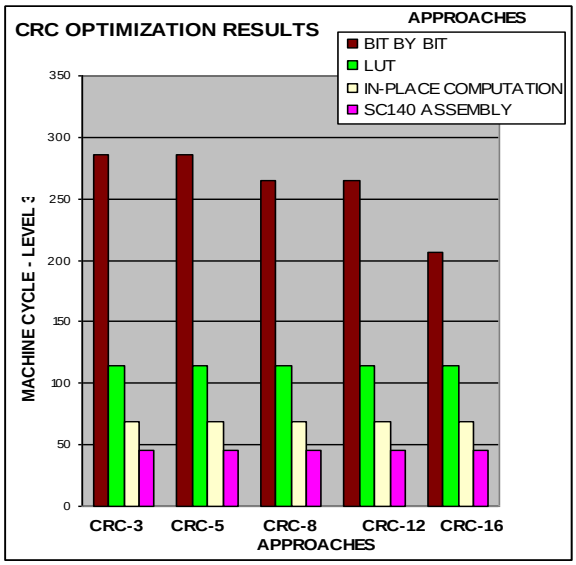
CRC Variants	*Optimization Level*	*Cycles (release 0.01) Bit by bit (BYB)*	*Cycles (release 0.02) (LUT)*	*Cycles (release 0.03) In-place Computation (IPC-LUT)*	*Assembly Implementation using SC140*
CrcEnc_3_5 (POLY_LEN=3)	0	2511	914	-	-
	3	**286**	**114**	**69**	**45**
CrcEnc_3_5 (POLY_LEN=5)	0	2473	914	-	-
	3	**286**	**114**	**69**	**45**
CrcEnc_8_12 (POLY_LEN=8)	0	2325	914	-	-
	3	**265**	**114**	**69**	**45**
CrcEnc_8_12 (POLY_LEN=12)	0	2235	914	-	-
	3	**265**	**114**	**69**	**45**
CrcEnc_16 (POLY_LEN=16	0	1783	914	-	-
	3	**206**	**114**	**69**	**45**

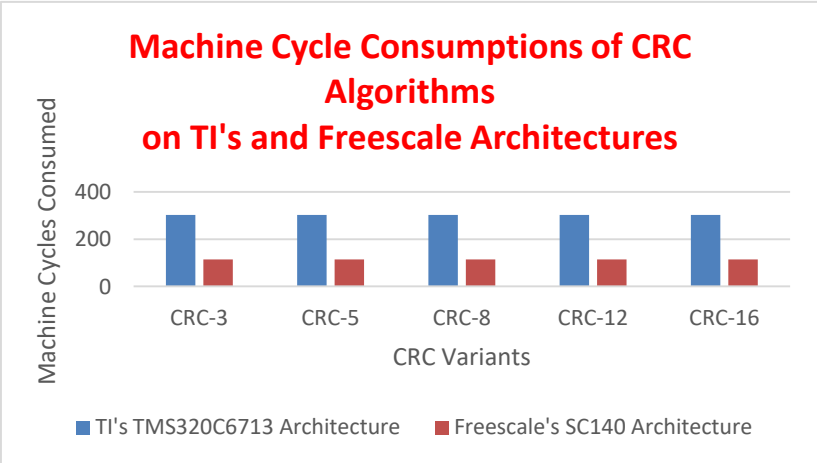
In LUT method, CRC for all bytes are precomputed using BYB (Total CRC's 256) and stored in the memory and then the 8-bits of message which are to be encoded is used as an index to get the corresponding CRC values from stored memory and appended to it. In our proposed IPC-LUT approach, in place computation is used to reduce the memory reference pointer and hence reduces machine cycle requirements drastically (Machine cycle reduces from 114 to 69, around 40% reduction is achieved).

- In place computation implementation is to make the processing in-place, i.e. no separate output buffer. The computed CRC is appended to the end of the input buffer. In LUT, separate input and output buffers were used.
  UINT1 *u1InpPtr; // Pointer to Input buffer; UINT1 *u1OutPtr; // Pointer to Output buffer
  Copying input to output buffer and the indexing the memory for precomputed CRC values was creating lot of memory read and write operation. This was consuming huge machine cycles. In proposed IPC-LUT, we have reduced these memory references using in place computation. This has also reduced the memory requirement to some extent. Removed following members of the *stCrc* structure
  - output buffer pointer
  - output length
- In LUT approach, the members of structure *stCrc* were being directly accessed from inside the loop, resulting in extra memory reads. For IPC-LUT, pointer to the input buffer and pointer to the LUT are stored in local variables initially. The local variables are then accessed inside the loop wherever required. Due to this reduction in machine cycle count was observed. Finally, IPC-LUT approach and Data Arithmetic & Logical Units (DALU) available in SC140 architecture are properly utilized to develop highly optimized assembly routine for CRC computation. CRC kernel takes only 45 machine cycles for computation.

Table 3. Comparison of machine cycle consumption for LUT approach on Freescale's SC140 and TI's TMS320C6713 architectures

CRC Calculation using Look Up Table (LUT) Approach	**Machine Cycles Consumed**				
	CRC-3	**CRC-5**	**CRC-8**	**CRC-12**	**CRC-16**
SC140 Architecture	114	114	114	114	114
TMS320C6713 Architecture	303	303	303	303	303





The same code is executed on two different architectures for understanding the profiler performance. From , it is clear that, machine cycle count drastically reduces on SC140 architecture. For designing the embedded applications, capability of processor architecture is equally important for porting the optimized algorithm.

5 Low PAPR Discrete Sequences

The PAPR is defined as the ratio between the maximum instantaneous power and the average power, defined by [8]-[10]

$$\text{PAPR} = \frac{P_{peak}}{P_{average}}$$

PAPR can be measured either in continuous time or in discrete time [10].

$$\text{PAPR}=\frac{[max|x_n|^2]}{E[|x_n|^2]}$$

Table 4. PAPR values of all possible data blocks for an OFDM signal with four Subcarriers and BPSK modulation

Data Block (X)	PAPR (dB)	Data Block(X)	PAPR (dB)
[1,1,1,1]	6.0	[-1,1,1,1]	2.3
[1,1,1,-1]	2.3	[-1,1,1,-1]	3.7
[1,1,-1,1]	2.3	[-1,1,-1,1]	6.0
[1,1,-1,-1]	3.7	[-1,1,-1,-1]	2.3
[1,-1,1,1]	2.3	[-1,-1,1,1]	3.7
[1,-1,1,-1]	6.0	[-1,-1,1,-1]	2.3
[1,-1,-1,1]	3.7	[-1,-1,-1,1]	2.3
[1,-1,-1,-1]	2.3	[-1,-1,-1,-1]	6.0

From above table, it is clear that high value of PAPR (6 dB) exist for sequences [1, 1, 1, 1], [1, -1, 1, - 1], [-1, 1, -1, 1] and [-1,-1,-1,-1]. These sequences have specific patterns of bits like continuous 1 and -1, alternate 1 and -1. There is no randomness present in these sequences. Hence more randomness in the sequence, indicates less PAPR value. Now the question is how to determine the randomness present in the sequence. We have worked on following testing criteria that provides good measure of randomness [13]:

Test 1: The first approach to judge the randomness of the sequence is the bit occurrence test. If number of ones is very close to number of zeros then that binary sequence exhibits good randomness.

Let sequences be p= p1, p2, p3,…….., pn then

$$A_n = (2p_1 - 1) + (2p_2 - 1) + \cdots + (2p_n - 1)$$

A_n measures the difference between number of ones and zeros. If A_n is small, then sequence has good randomness.

Test 2: Second test is to determine the number of "Runs" present in the sequence. Run is subsequence with continuous o or 1. If number of runs is approximately equals to half of sequence length then that sequence has good randomness.

$$B_n = \sum_{k=1}^{k=n-1} U_k \ . \ U_{k+1}$$

Where, $U_k = (2p_k - 1)$ and k=1, 2,…..n

Smaller the$\boldsymbol{B_n}$ **,** nearer the number of runs approximate to length n/2. However, consider the sequence 110011001100. It's a short cycle sequence with very high PAPR. But for this sequence number of runs equals to half of sequence length and number of ones equals to number of zeros. Hence if we apply first two test on this sequence then our decision will be wrong. Therefore, the decision criteria should be amended.

Test 3: To find aperiodic autocorrelation of sequence [13] R (i). But $\boldsymbol{B_n}$ is nothing but R (i) calculated at i=1. R (i) must be very small to have good randomness in the sequence.

$$C_n = R(2) = \sum_{k=1}^{k=n-2} U_k \ . \ U_{k+2}$$

Hence sum of A_n, B_n and C_n must be small, to have good randomness in the sequence.

Now our decision criteria for generating the low value PAPR sequences is as follows:

$$Q = {A_n}^2 + {B_n}^2 + {C_n}^2$$

Forward Error Correction (FEC) generates the sequences. But these sequences should have low value of Peak to Average Power Ratio (PAPR).Above decision criteria can be used for monitoring the sequences that exhibits low PAPR. We transmits only those sequences or codewords that has low PAPR to avoid the distortions caused due non linearity in power amplifier. Low PAPR sequences consumes less power. Thus system becomes power efficient.

6 Conclusions

CRC computation using BYB, LUT and our proposed IPC-LUT approaches are implemented on Freescale SC140 architecture and compared the machine cycle counts. Our results shows that, CRC computation using proposed IPC-LUT approach drastically reduces machine cycle count compared to conventional BYB and LUT approach. We have also developed optimized inline assembly code for CRC on SC140 architecture using our proposed IPC-LUT approach and tested for machine cycle count. It takes only 45 machine cycles. Result shows that Freescale Star Core SC140 architecture provides better optimization compared to TMS320C6713 architecture. Overall, we hope that our results provide embedded application engineers with better trade off information of machine cycle consumption, architectural profiling and power level constraints for discrete sequences.

References

[1] Mark Woh1, Sangwon Seo1, Hyunseok Lee1, Yuan Lin 1, Scott Mahlke1, Trevor Mudge1,Chaitali hakrabarti2, and Krisztian Flautner3 for "Next Generation Challenge for Software Defined Radio", 1 University of Michigan - Ann Arbor, Ann Arbor MI, USA, 2 Arizona State University, Tempe, AZ, USA, 3 ARM Ltd., Cambridge, UK, Springer-Verlag Berlin Heidelberg 2007.

[2] Lee H., Lin, Y., Harel, Y., Woh, M., Mahlke, S.A., Mudge, T.N., Flautner, K. "Software defined radio - a high performance embedded challenge" Conte, T., Navarro, N., Hwu, W.-m.W., Valero, M., Ungerer, T. (eds.) HiPEAC 2005. LNCS, vol. 3793, pp. 6–26. Springer, Heidelberg (2005).

[3] Ming Jiang, Member IEEE, and Lajos Hanzo, Fellow IEEE for "Multi User MIMO-OFDM for Next Generation Wireless Systems", Proceedings of the IEEE | Vol. 95, No. 7, July 2007.

[4] Luis Munoz, Marta García, Johnny Choque, Ramón Agüero, University of Cantabria Petri Mähönen, University of Oulu for "Optimizing Internet Flows over IEEE 802.11b Wireless Local Area Networks: A Performance-Enhancing Proxy Based on Forward Error Correction", IEEE Communication Magzine, December 2001.

[5] Ramabadran T V, Gaitonde S S , "A tutorial on CRC computations", IEEE Micro, 1988, 8(4): 62-75.

[6] Justin Ray, Philip Koopman, "Efficient High Hamming Distance CRC's for Embedded Networks", Carnegie Mellon University, Research Showcase, 2006.

[7] Gam D. Nguyen, "Error Detection Codes: Algorithms and Fast Implementations", IEEE transactions on Computer, Vol. 54, No. 1, January 2005.

[8] T A Wilkinson, A E Jones, "Minimisation of the Peak to Mean Envelope Power Ratio of Multicarrier Transmission Schemes by Block Coding", HP Laboratories, UK and Motorola, UK, 0-7803-2742-X/95, 1995 IEEE.

[9] Sen Hung Wang, Chih Peng Li, Kuan Chou Lee, and Hsuan-Jung Su, "A Novel Low Complexity Precoded OFDM System with Reduced PAPR", IEEE Transactions on Signal Processing, Vol. 63, No. 6, March 15, 2015.

[10] Yang Jie, Chen Lie, Liu Quan and Chen De, " A Modified Selective Mapping Technique to reduce the Peak to Average Power Ratio of OFDM Signal", IEEE Transactions on Consumer Electronics, Vol. 53, No. 3, August-2007.

[11] Third Generation Partnership Project Technical Specifications for "3GPP TS 36.211", V8.1.0 (2007-11).

[12] Gene Frantz, Texas Instrument, Principal Fellow for "Comparing Fixed and Floating Point DSP's".

[13] Dilip V. Sarwate, "Bounds on Crosscorrelation and Autocorrelation of Sequences", IEEE Transactions on Information Theory, VOL.IT-25, NO. 6, NOVEMBER1 979.