INT J COMPUT COMMUN, ISSN 1841-9836
9(2):187-200, April, 2014.

VLSI Architecture for High Performance 3GPP (De)Interleaver for Turbo Codes

J.M. Mathana, S. Badrinarayanan, R. Rani Hemamalini

J. Magdalene Mathana
Electronics and Communication Dept,
S.A Engineering College, Chennai, India
E-mail: jm.mathana@gmail.com

S. Badrinarayanan
Vinayaka Missions University, Salem, India
E-mail: badri22.jm@gmail.com,

R. Rani Hemamalini
Elect. and Comm. Dept,
St.Peter's College of Engg and Tech, Chennai, India
E-mail: ranihema@yahoo.com

Abstract: Interleaving along with error correction coding is an effective way to deal with different types of error in digital data communication. Error burst due to multipath fading and from other sources in a digital channel may be effectively combated by interleaving. Normally the interleaver / deinterleaver pair is often designed as reconfigurable architectures able to deal with requirements of large data length variability found in the newest communication standards. In this work reconfigurable interleaver architecture for the turbo decoder in 3rd Generation Partnership Project (3GPP) standard is presented. The interleaver is a key component of radio communication systems. Using conventional design methods, it consumes a large part of silicon area in the design of turbo encoder and decoder. The proposed interleaver utilizes the algorithmic level hardware simplifications and generates 100 manage the ow of data streams to achieve very low cost solution. The proposed technique reduces consumption of FPGA resources to a large extent compared with existing state-of-the-art interleaver for turbo codes. The proposed architecture con- sumes only 4856 logic elements by hardware optimization.
Keywords: 3GPP, interleaver, reconfigurable, turbo codes.

1 Introduction

Turbo codes [1] have been shown to provide superior error performance amidst hostile transmission media. 3rd generation systems based on the 3GPP air interface standard [2] must support turbo coding at data rates of between 384 kbps to 2Mbps, while maintaining low mobile station power. A turbo code is formed from the parallel concatenation of two codes separated by an interleaver [3] .The generic design of a turbo code is depicted in Figure 1. The two encoders used are normally identical. The code is in a systematic form, i.e. the input bits also occur in the output. The Turbo decoder consists of two elementary decoders in a serial concatenation scheme. Since soft decoding performs better than hard decoding, the first decoder provides a weighted soft decision in the form of A Posteriori Probabilities (APPs) to the second decoder. The decoding proceeds in an iterative fashion as illustrated in Figure.2 [4]. The interleaver reads the bits in a pseudo-random order. The choice of the interleaver is a crucial part in the turbo code design. The task of the interleaver is to "scramble" bits in a pseudo-random, albeit predetermined fashion. This serves two purposes. Firstly, if the input to the second encoder is interleaved, its output is usually quite different from the output of the first encoder. This means that even if one of the output code words has low weight, the other usually does not, and there is a smaller chance of producing an output with very low weight. Higher weight is beneficial for the performance of the decoder. Secondly, since the code is a parallel concatenation of two codes, the divide-and-conquer strategy can be employed for decoding. If the input to the second decoder is scrambled, also its output will be different or "uncorrelated" from the output of the first encoder. This means that the corresponding two decoders will gain more from information exchange [5].

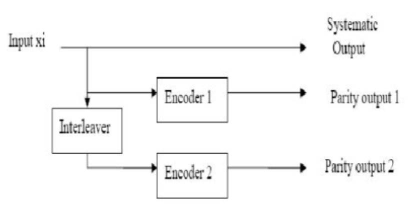


Figure 1: Turbo Encoder

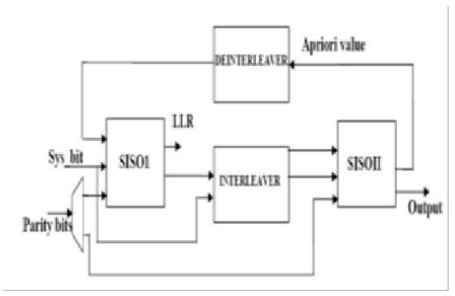


Figure 2: Turbo Decoder

A flexible and reconfigurable interleaver architecture for multimode communication environment is presented in [6]. The presented hardware in [6] enables the mapping of vital types of interleavers including multiple block interleavers and convolutional interleaver onto a single architecture. A parallel interleaver design that support for high throughput is presented in [7].In the proposed work, circuit level VLSI optimization approach is used to reduce the overall computation complexity and provides 100 % interleaved paths.

The paper is organized as follows. In section 2, a review of 3GPP algorithm is discussed. Section 3 describes the proposed reconfigurable architecture of interleaver/deinterleaver. The controller for the proposed reconfigurable interleaver architecture operated in two phases is explained in section 4. Section 5 describes about the address generation and section 6 describes about exception handling. In section 7, simulation and synthesis results are reported and section 8 concludes the paper.

2 Interleaver algorithm

The 3GPP-defined turbo code interleaver algorithm [8] comprises a series of mathematically complex processes that map an input sequence of length K (40 - 5114) to a scrambled sequence. The algorithm can be summarized as follows.

Block size 'K'denotes the number of bits input to turbo code internal interleaver and takes one of the value from 40 to 5114.

Determine the number of rows 'R' of the rectangular matrix such that,

R= 5; if (40<K<159)
R=10; if((160<K<200)or(481<K<530))
R=20 ;if (K= any other value)
where rows are numbered form 0 to R-1.

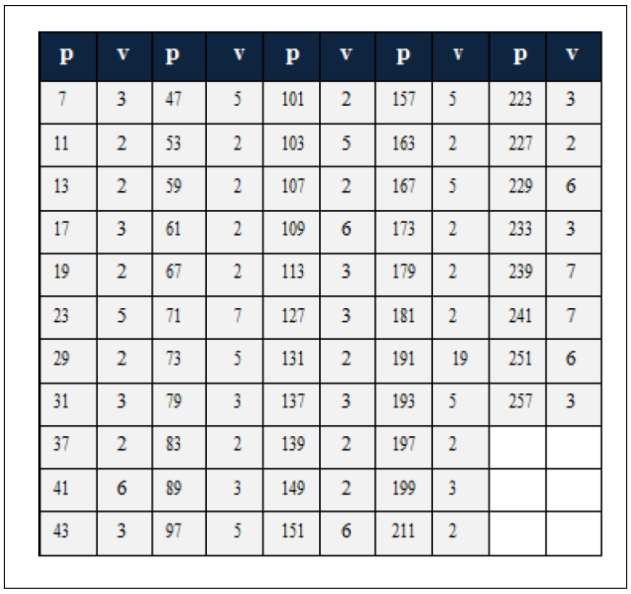
Determine the prime number 'p' to be used in intra-row permutations and the number of columns 'C' of the rectangular matrix as
if (481< K <530)
then p= 53 and C = p

else

Find the minimum prime number p from table 2 in [6] such that K < (Rx(p-1)) and determine C Such that
C = p-1; if K< (R (p-1))
C = p; if R (p-1) $< K <$ R p)
C=p+1; if (R p) < K

For every prime number p there exists a primitive root v.
Construct the base sequence S(j) for intra row permutation such that
S(j) = (v*S(j-1))mod (p) and S(0) = 1 where j=1,2,...,(p-2).
Determine the prime integers in the sequence q(i) such that
g.c.d(q,p-1) = 1, q(i)> 6 and q(i)> q(i-1) for i=1,2,.....R-1.
Permute the sequence q(i)to make the sequence r(i) such that
r(i) = T(q(i)), i = 0,1,...,R-1. where T(i) is a simple indexing transform.
It is defined by the standard and shown in Table 1.

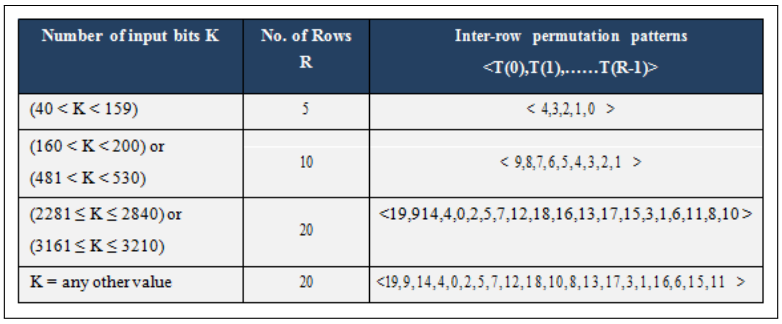
Table 1 List of prime number p and associated primitive root 'v'



p	v	p	v	p	v	p	v	p	v
7	3	47	5	101	2	157	5	223	3
11	2	53	2	103	5	163	2	227	2
13	2	59	2	107	2	167	5	229	6
17	3	61	2	109	6	173	2	233	3
19	2	67	2	113	3	179	2	239	7
23	5	71	7	127	3	181	2	241	7
29	2	73	5	131	2	191	19	251	6
31	3	79	3	137	3	193	5	257	3
37	2	83	2	139	2	197	2		
41	6	89	3	149	2	199	3		
43	3	97	5	151	6	211	2		

Perform the intra row permutation
if C = p , U(i,j)=S[(j*r(i))mod(p-1)]   where U(i,p-1) = 0
if C= p+1,U(i,j)=S[(j*r(i)mod(p-1)]   where U(i,p-1) = 0; and U(i,p) = p
and   if K= R C, then exchange U(R-1,0) with u(R-1,p)
if C = p-1, U(i,j)=S[(j*r(i))mod(p-1)]-1 where i = 0,1,...,R-1
and j = 0,1,..,p-2.

Table 2 Inter-Row Permutation Patterns



Number of input bits K	No. of Rows R	Inter-row permutation patterns <T(0),T(1),......T(R-1)>
$(40 < K < 159)$	5	< 4,3,2,1,0 >
$(160 < K < 200)$ or $(481 < K < 530)$	10	< 9,8,7,6,5,4,3,2,1 >
$(2281 \le K \le 2840)$ or $(3161 \le K \le 3210)$	20	<19,914,4,0,2,5,7,12,18,16,13,17,15,3,1,6,11,8,10>
K = any other value	20	<19,9,14,4,0,2,5,7,12,18,10,8,13,17,3,1,16,6,15,11 >

Perform the inter row permutation for the rectangular matrix based on the pattern T(i).It is denoted as U(i)
U(i) = U(T(i)), where i=0,1,..,R-1.
Read out the addresses column wise.

Looking at the interleaver algorithm in 3GPP standard, the computationally intensive functions found are Modulo Computation, Intra-row and Inter-row Permutations, Multipliers, Finding least prime integers, and Computing greatest common divisor. Some of these complex functions are implemented using the support from ROM, while the others are simplified to reduce the hardware usage.

The interleaving process can be easily separated into two main phases. The two phases are Pre-computation phase and Execution phase. As different block sizes has different interleaving patterns, thus every time some pre-computation is needed while changing the block size 'K'. The only parameter applied to interleaver is the block size (K) and rest of the parameters is computed by the hardware itself.

2.1 Pre-computation Phase

Pre-computation phase involves pre-computing the various parameters required for the interleaving process. It has to be performed each time the change in the block size occurs, that means for a fixed block size K, these operations have to be performed only once.

2.2 Execution phase

Execution phase calculates the intra row permutations Ui,j, and the interleaved address i_addr .Write or Read the data bits (in/ from a data RAM) can be taken place depending upon whether interleaving or deinterleaving is performed of addresses could also be computed and saved in the memory only.

For each data block, the execution phase has to be performed. The set of addresses could also be computed and saved in the memory only once.This approach permits to reduce the implementation size of Interleaving / Deinterleaving algorithm as compared to the memory or look up table (LUT) based approach, where the interleaving addresses are saved in memory. Thus execution phase performs basic permutations which form the interleaving process. The basic permutations performed in this phase are illustrated in fig.4.

3 Proposed Architecture of Reconfigurable Interleaver

The main objective of the proposed work is to device an architecture capable of managing every one of the 5074 different block sizes of data ranging from 40 to 5114 defined in the 3GPP

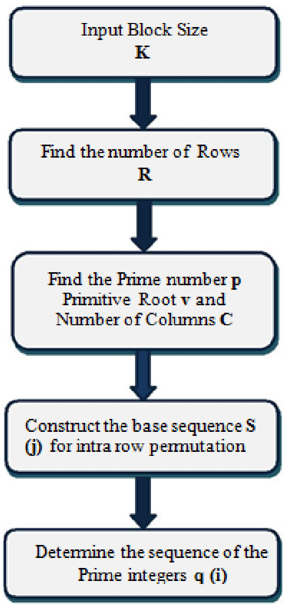


Figure 3: Flow graph for Pre-computation Phase

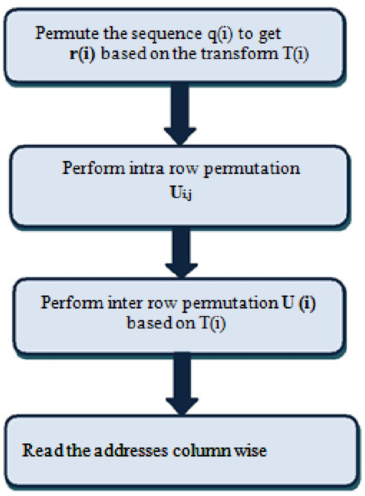

Permute the sequence q(i) to get
r(i) based on the transform T(i)
Perform intra row permutation
Uij
Perform inter row permutation U (i)
based on T(i)
Read the addresses column wise

Figure 4: Flow graph for Execution Phase

standard, while maintaining low hardware complexity. The proposed interleaver is said to be reconfigurable, since it adapts or varies its interleaving architecture depending upon the block size 'K' of the input rectangular matrix. The reconfigurable interleaver accepts only the block size of the rectangular matrix as the input. The rest of the parameters such as the number of rows, columns and the base sequence for permutation (both intra row and inter row permutation) are calculated by the reconfigurable architecture for interleaver address generation. The block diagram of proposed reconfigurable interleaver/deinterleaver is shown in Figure 5. It consist of modulo-computation block, Multiplication addition and comparison hardware, Controller circuit , I/O RAM, Data RAM, Exceptional logic block, S(j) RAM and LUTs etc.

3.1 Precomputation of Parameters

The computation of parameters in pre-computation for 3GPP is discussed in detail by Blakley [9]. Some parameters are computed using look up tables while the others need some close loop or recursive computations. First, the row value 'R' with logical functions is calculated. Parameters like prime number 'p' and primitive root 'v' are stored as a pair in a look up table just like in the standard and LUT is read via a counter generated by the controller based on equation (1).

$$K \leq R * (p + 1) \rightarrow (K - R) \leq (R * p) \tag{1}$$

In order to perform the operation in equation (1) the architecture given by Blakley is used in the proposed work. It is shown in Figure 6.The same hardware architecture is used in execution phase also. For computation of 'p' and 'C' operations such as the multiplication, addition and comparison are required. It is controlled by the flags generated by the controller. Once 'p' and 'v' are obtained then numbers of columns have to be determined.

The three possible column values are p-1, p or p+1 and always the following condition shown by the equation (5.2) should be satisfied.

$$(R * C) \geq K \tag{2}$$

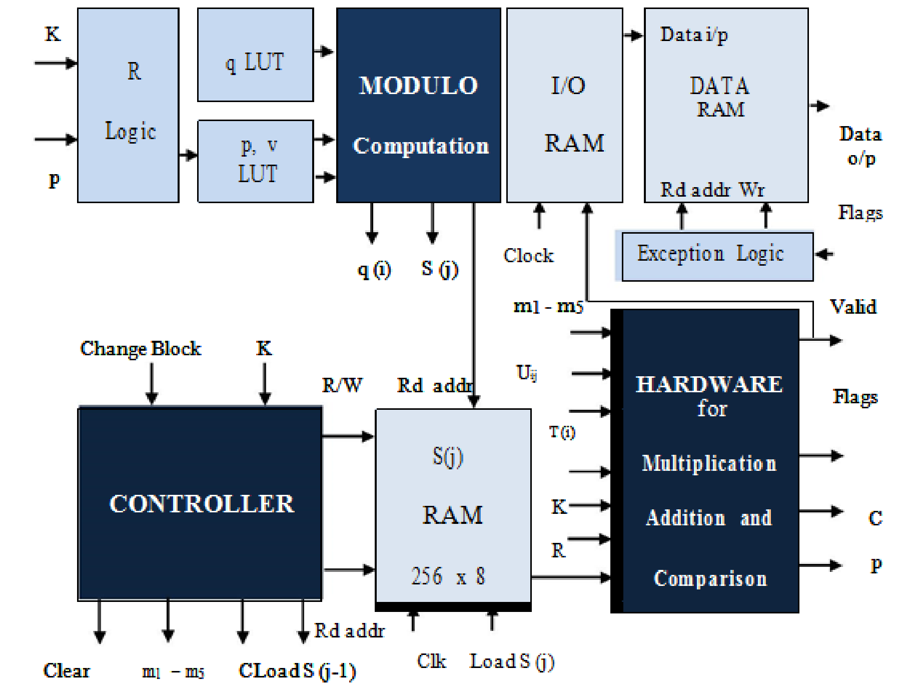


Figure 5: Block Diagram of Proposed Interleaver/Deinterleaver

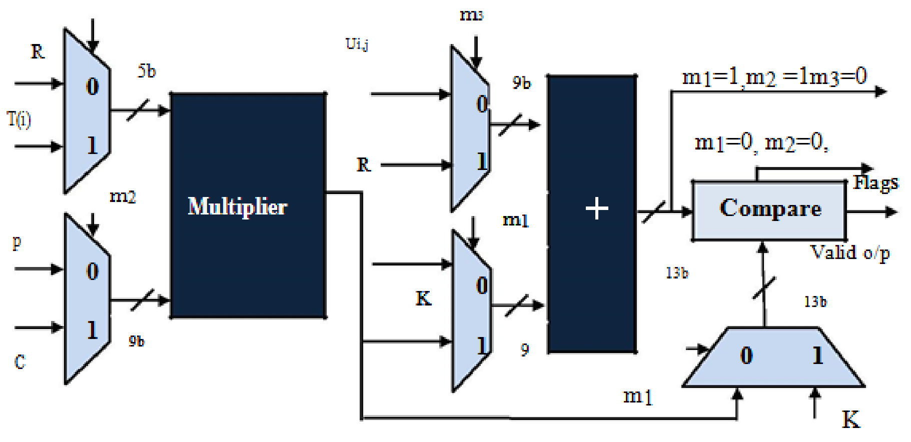


Figure 6: Hardware for Multiplication, Addition and Comparison

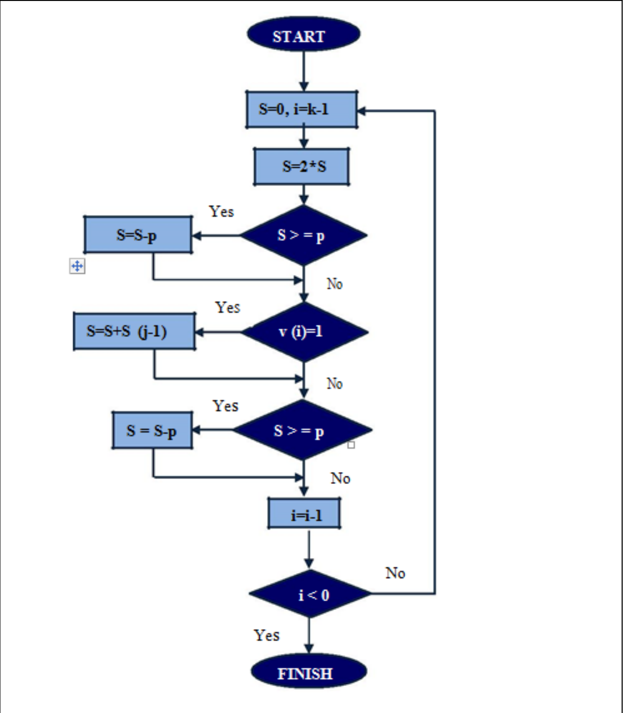


Figure 7: Flow Diagram for the Calculation of S(j)= (v ×S (j-1)) Mod p

The computation of intra-row permutation pattern S(j) required modulo computation. Modulo function is computed iteratively using the Interleaved Modulo Multiplication Algorithm [9].It is an iterative numerical algorithm based on additions, shifts, comparisons and bit retrieval. In order to calculate the S(j) sequence, it is mandatory to perform the modulo operation. A flow diagram depicting this algorithm is shown in Figure 7.

The hardware for the computation of S (j) is shown in Figure 8. This architecture uses two adders, a multiplier and a subtractor to compute the S (j) values. The S array is computed as follows (3) with S (0) = 1.

$$S(j) = (v * S(j-1))mod(p) \tag{3}$$

Where j =1, 2... (p-2)

From the above equation it is known that, primitive root 'v' is required to compute S(j), which is 5 bits. So maximum of 5 iterations are needed to compute one modulo multiplication. With this architecture every S (j) is determined and writes in a RAM of 256 x 10 bits. The size of the RAM depends upon the value of 'p'. The number of cycles needed to calculate every S (j) depends on v, which can take only six different values 2, 3,5,6,7 and 19. For v = 2 and v = 3 only one iteration is required for v = 5, v = 6 and v =7 two iterations and for v =19 four iterations are required.

According to the 3GPP algorithm, the next step is calculation of the prime sequences q (i). It can be noticed that the qi sequences [10] are almost the same in most cases and they only differ by one or two elements from other sequences. Based on this observation, sub groups of q (i) sequence are placed into a ROM and then choose one according to the 'p' value [11]. In the proposed work, instead of finding the least prime number sequence q (i), q(i) mod (p-1) is determined. This gives the benefit of computing the RAM address recursively and avoiding computation of modulo function. This idea is introduced by Shin and Park [12] and later used by Wang and Li [13]. According to the equation (4) q (i) is calculated.

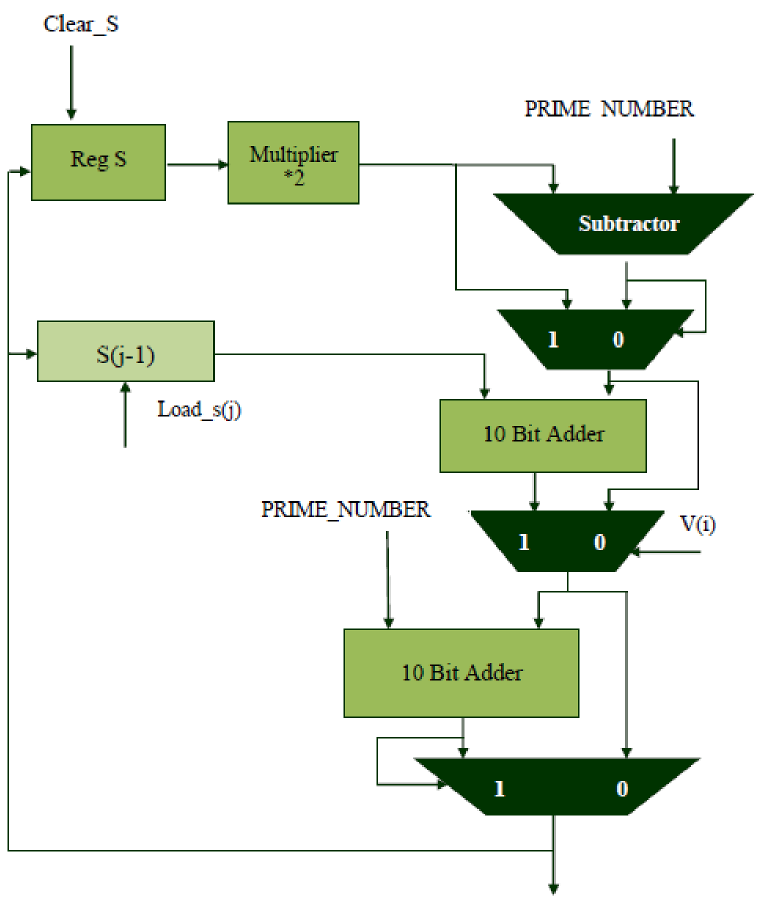


Figure 8: Architecture for Performing [v ×S (j-1)] Mod p

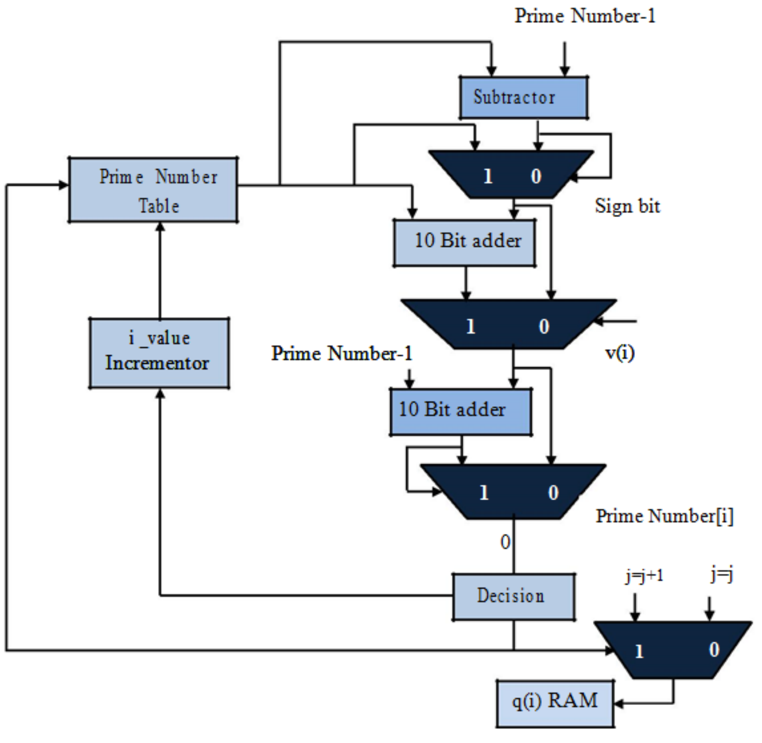


Figure 9: Architecture for Performing q(i)

$$q(i) = q(i) mod p - 1 \tag{4}$$

where i = 0,1,2,3,...p-1

The architecture used to perform q(i) is shown in Figure 9. In the proposed work, the architecture of S (j) is used with some alterations to determine q mod (p-1). It is calculated in such a way that the following conditions have to be satisfied.

g c d(q, p-1) = 1, q(i) > 6 and q(i) > q(i-1) for i = 1,2,.....R-1.

g c d is the Greatest Common Divisor. The selected prime number from the prime number table should be 6 and g c d (q, p-1) = 1.If the above condition is satisfied then the value of i = i+1, j = j+1 and the next prime number is selected. The selected prime number becomes one of the member of q(i) and the value of 'j' is also incremented by1.If the condition is not satisfied then i=i+1,j=j.

3.2 Execution Phase Computation

After completing the pre-computation phase, the controller is set in execution phase and the hardware is configured to perform run time computations for the generation of the interleaved addresses.RAM address are computed using the hardware shown in Figure 6. The recursive function used to compute the RAM address and multiply?add function to compute the final interleaved address are as follows:

$$Ram_a dr(i, j) = [Ram_a dr(i, j-1)\ q mod\ (i)] mod(p-1) \tag{5}$$

It can be seen that computing the RAM address using qmod (p-1) instead of q helps to avoid the full computation of modulo multiplication. After computing the RAM Address, the final interleaved address is computed by the multiply?add function. The interleaved addresses are obtained by performing the inter row T (i) and intra row Ui,j permutations in executing phase, where T(i) is stored in a LUT while the Ui,j is stored in a RAM. With these parameters and the using the equation (6), the interleaving addresses i_addr are finally generated.

$$i_a ddr\ ((i \times C) + j) = (C \times T(i)) + Ui, j \tag{6}$$

with i = 0,1,...,R-1; j = 0,1,...,C-1

[C×T(i)] is computed by the architecture shown in Figure 6.The result of this computation points to the first element of each of the 'R' rows in the rectangular matrix, and then by adding Ui,j with the same architecture, a displacement along every row is obtained. Ui,j is obtained by calculating modulo operation (j × r(i)) mod (p-1).It has the same form as (v × S(j-1))mod p in the pre-computation phase. Since the computation of S (j) and Ui,j are performed at different phases, the same architecture is used with some modifications. Since the same architecture is used in both the phases, hardware complexity is reduced.

3.3 Modulo Computation

One of the important blocks in the proposed reconfigurable architecture is "Modulo Computation". This block plays a crucial role both in execution phase and in pre-computation phase.

Figure 10 shows the architecture of modulo computation introduced by Rizwan Asghar and Dake [10]. The main block is depicted in Figure 10 by blue colour. It is separated and shown in Figure 11a. In pre-computation phase, this architecture works 100% efficiently. But in execution

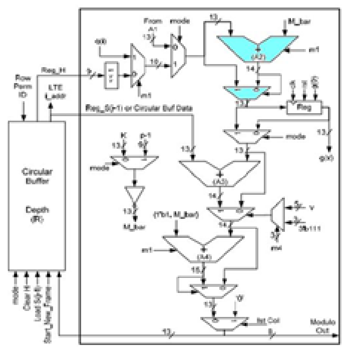


Figure 10: Modulo Computation

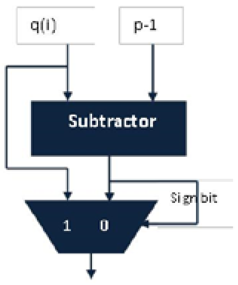


Figure 11a: q(i) Mod p-1

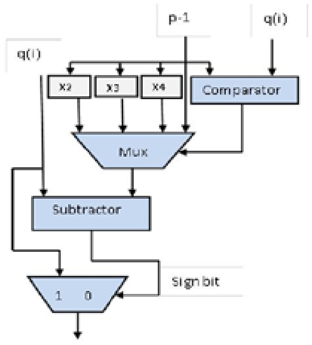


Figure 11b: q(i) mod p-1 with Modification

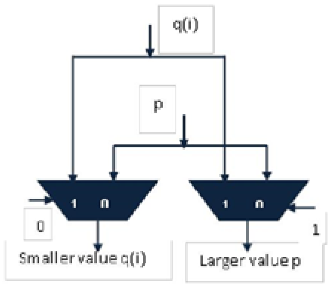


Figure 12: Modified Architecture

phase, its efficiency is only 88%. The problem will occur when q (i) > 2(p-1)[10]. This problem is nullified by performing subtractions like q(i) - (p-1), q(i) - 2(p-1), q(i) - 3(p-1) and q(i) - 4(p-1) instead of q(i) - (p-1) [11]. It is illustrated in Figure 11b. In the proposed work, this structure is further modified as shown in Figure 12.

In order to nullify the problem, always q(i) must be less than p. In the proposed work q (i) is compared with p. After comparison the small value is assigned to q (i) and the large value is assigned to p .Hence in the proposed work, during the execution phase also modulo computation block works 100% efficiently. There are three multipliers are used.[11] to generate 2(p-1), 3(p-1) and 4(p-1). But the proposed modified architecture does not contain any multipliers; hence hardware complexity is reduced. This architecture performs all modulo operations required by the 3GPP standard. With this modification the obtained hardware architecture works properly in both the pre-computing and executing phase.

4 Control Finite State Machine

In order to synchronize every block in the architecture, a "Controller" block generates all control signals; its state machine diagram is shown in Figure 13.

The controller mainly works for the pre-computation phase of 3GPP. It configures different functions like multiply, add, compare and modulo computation hardware to find all the vital parameters R, C, p, v and S (j). Usually interleavers in the turbo coded system work with the same block size 'K' several times before changing it. When an interleaving operation is required the pre-computation phase is performed, where S (j) is calculated and placed in RAM. Then the execution phase starts where interleaved addresses are calculated continuously meanwhile 'K'

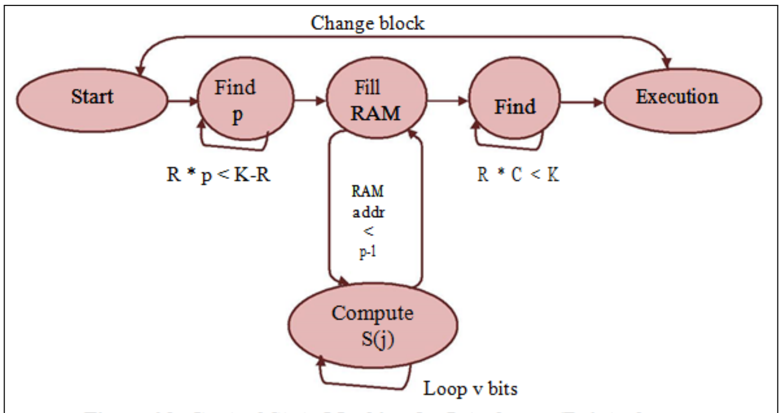


Figure 13: Control State Machine for Interleaver/Deinterleaver

does not change. The new pre-computation phase can be initiated by changing the block size 'K'.

5 Exception Handling

In the 3GPP interleaving algorithm, there are some exceptions that are reflected in the architecture. The final address is tagged valid or invalid using the comparator. This is called pruning of the interleaver and is needed for the case when interleaver block size is not exactly equal to R*C. For example, for K = 43 the rectangular matrix size is 5x10, and then there would be seven invalid addresses. Because of that, a valid address cannot be generated per clock cycle.

Interleavers presented by Shin and Park [12] Wang and Li[13], [11] and Carlos Sánchez [14] provides the interleaved addresses with valid or invalid tag. But in the proposed architecture, if the input data stream of any size between 40 and 5114, then the interleaver rearranges it according to the corresponding 3GPP interleaved path and provides the data stream even the exception is present. In order to manage data streams, a data RAM is used in the proposed design. When the proposed architecture is operated in interleaver mode, this RAM is used to hold data input when invalid addresses are generated and waiting for a valid address in order to write this data to DATA RAM. In deinterleaver mode, when invalid address is read from DATA RAM, the I/O RAM ignores this data, waiting for a valid data and then output this data.

6 Performance Analysis Of Reconfigurable Interleaver

The RTL code for the hardware blocks is written in Verilog and downloaded it to the FPGA Cyclone III from ALTERA. The RTL diagram of reconfigurable interleaver / deinterleaver is shown in Figure 14.

The number of clock cycles spent for pre-computation through is mentioned in Table 3. These cycles mainly depend on the value of p and v for a particular block size.

The maximum clock frequency of 69.91 MHz is shown in Figure 15.Table 4 summarizes comparison of the performance with existing state-of-the-art interleavers. As it can be seen from the Table 4, the proposed design occupies about 4586 out of 5136 logic elements.

It is worth mentioning that although the proposed design is bigger than others, it includes the hardware for exceptions handling, an I/O RAM and a DATA RAM as well as extra hardware to control them. In this way, the proposed architecture is capable of working both as an interleaver and as a de-interleaver.In this proposed interleaver/deinterleaver latency and throughput can

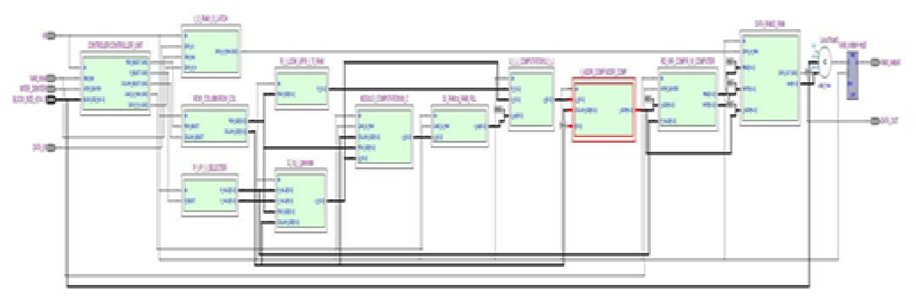
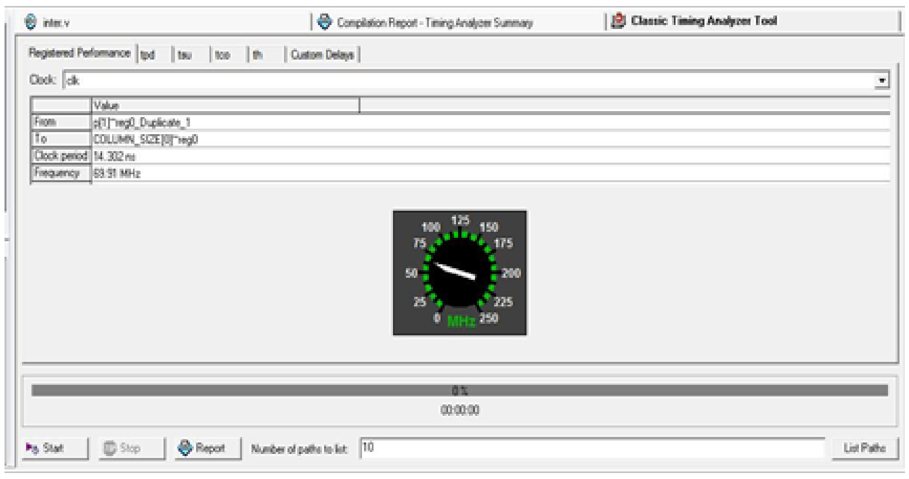


Figure 14 RTL Diagram of reconfigurable interleaver / deinterleaver

Table 3 Average Interleaved Address Rate for Different Block Size

Block Size	Pre-computation Cycle	Execution Phase Cycles/ Frame	Average valid address /cycle
40	27	40	1.000
41	27	42	0.970
2041	132	2050	0.995
4241	493	4410	0.961
4840	557	4840	1.000
5114	569	5120	0.998



Clock: clk

	Value
From	p[1]~reg0_Duplicate_1
To	COLUMN_SIZE[0]~reg0
Clock period	14.302 ns
Frequency	69.91 MHz

0 %

00:00:00

Number of paths to list: 10

Figure 15: Timing Analyzer Summary

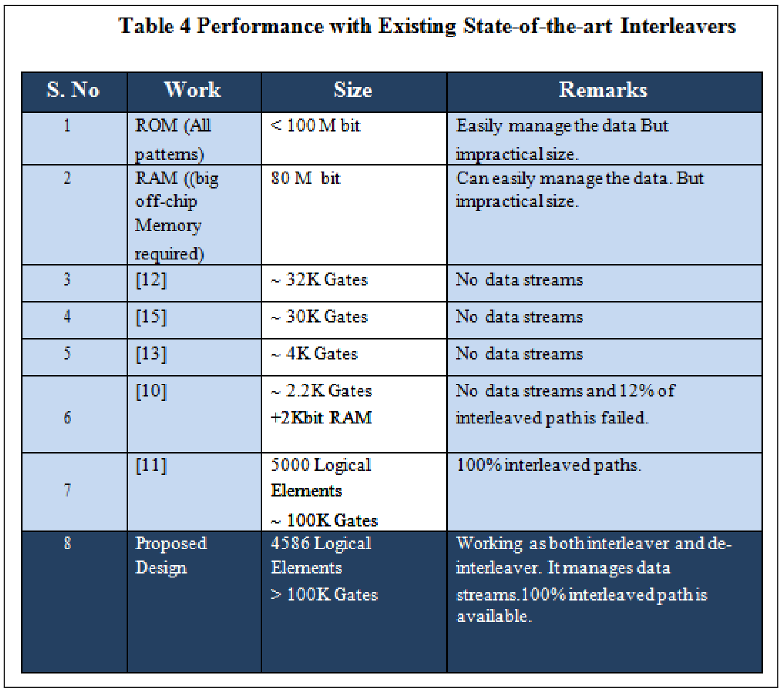


Table 4 Performance with Existing State-of-the-art Interleavers

S. No	Work	Size	Remarks
1	ROM (All patterns)	< 100 M bit	Easily manage the data But impractical size.
2	RAM ((big off-chip Memory required)	80 M bit	Can easily manage the data. But impractical size.
3	[12]	~ 32K Gates	No data streams
4	[15]	~ 30K Gates	No data streams
5	[13]	~ 4K Gates	No data streams
6	[10]	~ 2.2K Gates +2Kbit RAM	No data streams and 12% of interleaved path is failed.
7	[11]	5000 Logical Elements ~ 100K Gates	100% interleaved paths.
8	Proposed Design	4586 Logical Elements > 100K Gates	Working as both interleaver and de-interleaver. It manages data streams.100% interleaved path is available.

vary depending on the block size. The maximum block size is K=5114.The proposed architecture requires 5120 clock cycles which represents a realistic insignific ant overhead.

7 Conclusions and Future Work

In this paper a fully functional 3GPP Turbo code interleaver/ deinterleaver architecture that receives an input data stream of any size established by the 3GPP standard and delivers this stream interleaved or deinterleaved depending on the user requirements is presented. In this proposed design modulo computation is done by computer algorithm to calculate and took advantage of using the same hardware in pre-computation and execution phases by multiplexing it. Novel VLSI architectures have been introduced in this design. By adding RAMs for data handling and achieved a complete architecture that can perform interleaving/deinterleaving operations as required by the 3GPP standard for Turbo codes.One of the important parameter is pre-computation cycle cost. If the pre-computation cycle cost is less then, the design supports fast switching among different standards. Hence future work is to design reconfigurable interleaver which is suitable for a multimode environment.

Bibliography

[1] 3rd Generation Partnership Project (3GPP) TSG-RAN, "*Multiplexing and Channel coding,*" Release 4, Version 4.2.0, Sept. 2001.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima (1993), Near-Shannon Limit Error- Correcting Coding and Decoding : Turbo Codes, *Proceedings of ICC, Geneva,Switzerland*, 1064-1070.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima (2003), Near Shannon limit error correcting coding and decoding:Turbo codes. *Proceedings of the IEEE International Conference on Commun*, Geneva, Switzerland, May 2003.

[4] M Valenti (1999), *Iterative Detection and Decoding of Wireless Communications*, Ph.D thesis, Virginia Polytechnic and State University, July 1999.

[5] University of South Australia, Institute for Telecommunications Research, Turbo coding Research group, http://www.itr.unisa.edu.

[6] Rizwan Asghar and Dake Liu (2010), Multimode Flex- Interleaver Core for Baseband Processor Platform, *Journal of Computer Systems, Networks, and Communications*, 1-16.

[7] Rizwan Asghar and Dake Liu (2010), Towards radix4,parallel interleaver design to support high throughput turbo decoding for reconfigurability, *33rd IEEE SARNOFF symposium*, Princeton, New Jersey, USA, 1-5.

[8] *3rd Generation Partnership Project, Technical Specification Group Radio Access Network; Multiplexing and Channel Coding*,Release 6, 3GPP TS 25.212 v6.0.0 (2003-12).

[9] G.R.Blakley (1983), A Computer Algorithm for Calculating the Product A*B mod M, *IEEE Trans. On Comp*, 32(5):497-500.

[10] Rizwan Asghar and Dake Liu (2008), Very low cost Configurable Hardware Interleaver for 3G turbo decoding, *3rd International Conference on Information and Communications Technologies Theory to Applications*, ICTTA 2008, Damascus, Syria, 1-5.

[11] Hector Borrayo - Sandoval , R. Parra-Michel, Luis F.González-Pérez, Fernando Landeros Printzen Claudia Feregrino-Uribe (2009), Design and Implementation of a Configurable interleaver/deinterleaver for Turbo Codes in 3GPP Standard , In *Proc. of IEEE International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, 320- 325.

[12] M. Shin and I.-C. Park (2003), Processor - based turbo interleaver for multiple third generation wireless standard, *IEEE Commun. Lett.*, 7(5):210 - 212.

[13] Z. Wang and Q. Li (2007), Very low - complexity hardware interleaver for Turbo decoding, *IEEE Trans. on Circuits and Sys. - II*, 54(7):636 - 640, July 2007.

[14] Carlos R.Sanchez, R. Parra-Michel and M.E Guzmán- Renteria (2008), Design and implementation of a multi-standard interleaver for 802.11a, 802.11n, 802.16e & DV standards, *International Conference on Reconfigurabl Computing and FPGAs, ReConFig 2008*, Cancun, Mexico, 379 - 384.

[15] P.Ampadu and K. Kornegay (2003), An efficient hardware interleaver for 3G turbo decoding, *Proc. of Radio and Wireless Conference, RAWCON'03*, Boston, 199-201.