IRAQI Academic Scientific Journals

Al-Rafidain Engineering Journal (AREJ)
Vol. 25, No.1, June 2020, pp. 70-77

ISSN (Print) 1813-0526;
ISSN (Online) 2220-1270

Design Analysis of Turbo Decoder Based on One MAP Decoder Using High Level Synthesis Tool

Amer T. Ali
aamertali@yahoo.com

Dhafir A. Alneema
dhafir.abdulfattah@uomosul.edu.iq

Computer Engineering Department, Collage of Engineering, University of Mosul

Received: 14/3/2020 Accepted: 6/6/2020

ABSTRACT

High Level Synthesis (HLS) tool does not only simplify the designing operation and rapid prototyping but also allows the designers to explore large number of design's techniques such as parallelism, pipeline, memory partitioning and many other techniques. Turbo decoder based on Maximum APosterior Probability (MAP) algorithm is designed in this work using Vivado HLS. The normal turbo decoder with two MAP decoders were implemented with and without parallelism and proposed a new design of turbo decoder with one MAP decoder and it was designed with and without parallelism using different window technique in HLS tool which it is not explored previously. These designs were implemented for different frame size in this work. A comp-arison in latency and resource utilization where done and how a tradeoff done between these two parameters to reach the specific design that we need. The new design produces better results.

Keywords:

HLS tool; Turbo decoder; Latency; Resource utilization.

https://rengj.mosuljournals.com
Email: alrafidain_engjournal@umosul.edu.iq
===

1. INTRODUCTION

Communication systems are one of the most important elements of the requirements of modern times. The increasing demand for information and exchange of information among users of networks has become the most important necessities in contemporary daily life, therefore, attention has been given to this field and many work and research has been done to make the comm-unication systems highly capable to meet these requests. With the emergence and spread of wireless devices as well as the use of wireless networks, communication systems must do the best performance and increase the work in order to keep pace with this tremendous development in the field of information transmission [5].
One of the things that has been achieved in communication systems is to increase reliability specially in wireless communications because it transfers on noisy channel. Forward Error Correction (FEC), have been put in place to get the correct information for the recipient to increase reliability [7].
Forward Error Correction (FEC), is amath-ematical method that adds some known values to the message sent by the sender and the receiver of this message can received it correctly even in case of errors in the message during the transfer. In modern digital communication systems repre-sented in Fig. 1, the channel encoder and channel decoder are considered an important part of this system as a FEC [5].

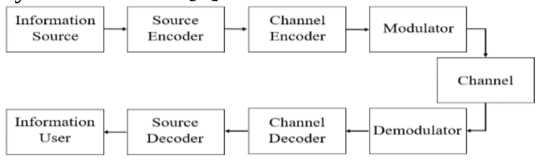

Fig. 1 General block diagram of digital communication systems

FEC has been introduced in many areas of wireless and space communications like 3^{rd}Generation Partnership Project Long Term Evolution (3GPP LTE), Global System for Mobile communications (GSM), IEEE Standard P802.16 also known as (WiMAX) Worldwide inter-operability for Microwave Access, Digital Video Broadcasting Satellite Services to Handheld (DVB-SH), Universal Mobile Telecommunication System (UMTS) and other applications [3].
Forward error correction has a long history so it has many types of coding. Each type has many different characteristics in possibilities, purposes and mathematical calculations like: The Repetition Code, The Parity Bit, The Hamming Code, Hadamard Codes, Golay Codes, Reed-Solomon Codes, Convolution Codes, low-density parity-check (LDPC), Turbo codes and many of them [15].
One of FEC types is the turbo code. The development and improvement of turbo encoder and turbo decoder is a wide area of research so there are many attempts to implement the turbo code. Many researches using Very high-speed integrated circuit Hardware Description Language (VHDL). Whereas Recently, the trend has turned to made implementations using High Level Synthesis (HLS) tools. Which is a tool for rapid prototyping and production of hardware designs with short developmental cycles in Register Transfer Level (RTL). It is based on widespread high-level language (HLL)C/C++. This will enable the user to design complex hardware designs [10].
The researchers in [1] using parallelism level 64 for frame size 2048-6144 and parallelism level 8 for frame size 256-2048 and using an interleaver proposed by them, and they conclude that the result of performance and latency that could use turbo code in future terrestrial broadcasting (TB) systems.
And in research [2] a comparison between turbo code, LDPC, polar code was done the result showed that the turbo code made the best performance than the others in error correction and

flexibility in using different block sizes and different code rates.
In paper [3] BER (Bit Error Rate) per-formance was investigated with multilevel of parallelism and for small frame size and concluded that self-concatenated convolutional code is better than the normal turbo code performance in BER.
Research in which HLS tools were verified as a means of producing prototypes and shortened development cycles needed to produce device designs at the RTL was done in [4]. A LDPC was used and their results showed that codecs that the using HLS tools, either as pipeline or as data flow designs, is able to reach the productivity of existing designs at the RTL.
While a master's thesis in [5] proposed by Conn, three types of turbo decoders were designed. These designs were implemented with different architecture using high level synthesis (HLS) tool that used in software defined radios (SDR).
46 articles were studied on the quality of the results and design efforts. The implemented designs were compared using HLS tools and RTL designs in [6] and they found that 40% of the cases studied proved that HLS tools equaled or outperformed RTL designs from In terms of performance and better use of resources, they also studied whether the size of the design affects performance quality, and they concluded that HLS tools are suitable for both large and small designs.
In paper [7] high level synthesis tool was used for implementing turbo decoder algorithms with exploration of HLS optimization using different directives for designing several archi-tecture designs of turbo decoders.
This work designs a normal turbo decoder with two decoders in iterative fashion in C++ language and make it fully parallel by unroll directive using Vivado HLS tool and make a comparative with a proposed turbo decoder with one decoder in iterative fashion and again made it fully parallels by same directive. These two decoders are decoding three different frame size of 108, 216 and 432 received bits, with two different windows, 9 decoded bits of 27 received bits and 36 decoded bits of 108 received bits and compare these designs in latency and resources utilization.

2. THE THEORETICAL BASES

This section talks about turbo code as one of forward error correction and formed by two stages: turbo encoder and turbo decoder. The next part is about HLS tools.

2.1. Turbo encoder

It is formed by two recursive systematic convolutional (RSC) encoder in parallel conca-tenation and the two encoders separated by an interleaver.
There are many researches on the (RSC) and how to design it but in this paper two memory

elements were used with recursive feedback and the generator polynomial is [16]:

$$G = (1, \frac{1+D^2}{1+D+D^2}) \quad \text{(p1)}$$

The main purpose of interleaver is to reduce the burst error to help for error correction. This paper uses a block interleaver. Block interleaver is look like two-dimensional array and inter the value of the input bits, bit by bit in rows way into this array and read these bits again, bit by bit in columns way. In the end of encoding each input bit well be decoded in three bits one is the systematic input and two parity bits. Fig. 2 explains the components of the turbo encoder.

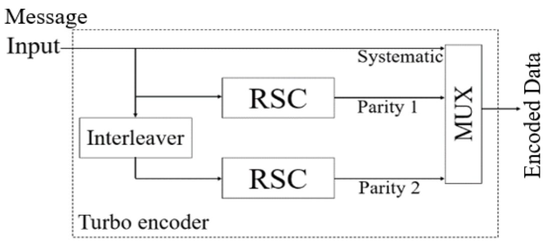

Fig. 2 Block diagram of turbo encoder.

2.2. Turbo decoder

All equations were mentioned in this section were taken from [5]. The second stage of turbo code is the turbo decoder, it is more complex than turbo encoder. The decoding technique based on trellis method for sot input soft output (SISO) algorithms in Fig. 3 turbo decoder is shown.

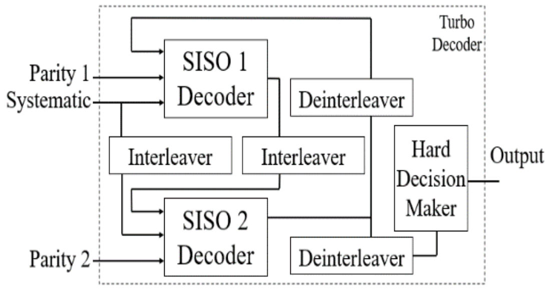

Fig. 3 Block diagram of turbo decoder.

Where SISO 1 and SISO 2 are the soft input soft output decoding algorithms. Hard decision maker is deciding whether the bit is zero or one depending on the value of decoder output, if the value is positive the bit will be one and if the value is negative the bit will be zero, this decision is taken after decoding the message.

The work of turbo decoder simply begins when SISO 1 decoder decode the massage and pass his opinion depending on the algorithms decode, to the SISO 2 decoder helping this decoder to make his decode and pass his opinion to the SISO 1 again in iterative fashion for suitable number of itera-tions. This opinion is named extrinsic information.

One of the (SISO) decoding algorithms is maximum a Posterior (MAP) or called (BCJR) algorithm [2]. MAP algorithm calcu-lates the logarithm ratio of the probability of bit that will be one over probability of bit that will be zero this ratio is named log likelihood ratio (LLR). The LLR is computed by the equation bellow:

$$LLR\ (u_k|y) = log\frac{P(u_k=1|y)}{P(u_k=0|y)} \quad (1)$$

Where P is the probability, u_k is the bit will be decoded and y is the bit was received from the channel.

After several mathematical analyzing and simplification, the equation will be like bellow:

$$LLR(u_k|y) = ln\frac{\sum_{r=1}\alpha_{k-1}(s')*\gamma(s',s)*\beta_k(s)}{\sum_{r=0}\alpha_{k-1}(s')*\gamma(s',s)*\beta_k(s)} \quad (2)$$

The numerator represents the multiplication of the values of alpha, gamma and beta for the bit equal one in transmits from one state to another in trellis. The denominator value is the same thing but for bit equal to zero, α_{k-1} is called the forward recursion and it is computed by the next equation:

$$\alpha_k(s) = \alpha_{k-1}(s') * \gamma(s', s) \quad (3)$$

$\alpha_k(s)$ is represented the alpha value for the next state, $\alpha_{k-1}(s')$ is the value of alpha for the current state and $\gamma(s', s)$ is the value of branch metric for branching from current state to next state. The value of alpha is calculated from the beginning to the end of trellis. The initial value of first alpha value is one for state zero and zero for the other states.

And β_k is called backward recursion and it is computed by the equation bellow:

$$\beta_{k-1}(s) = \beta_k(s') * \gamma(s', s) \quad (4)$$

Where $\beta_{k-1}(s)$ is the value of beta for the next state, $\beta_k(s')$ is the value of beta for current state and $\gamma(s', s)$ is the value of branch metric for branching from current state to next state. The value of beta is calculated from the end to the beginning of the trellis. The initial value of the

last beta value is one for state zero and zero for the others.
Finally, $\gamma(s',s)$ is the branch metric for branching from state to another state and it is computed by the following equation:

$$\gamma(s',s) = C_k \exp(\frac{1}{2}\begin{pmatrix} Lc\, y_k u_k + u_k\, Lu_k \\ +Lc \sum_{i=0}^{n} y_{k_i} x_{k_i} \end{pmatrix})(5)$$

Where C_k is a constant ignored because of the division, Lc is the channel reliability, y_k the bit received from the channel, u_k represent the trellis input, Lu_k represent the extrinsic information that came from the previous decoder, n equal two bits the systematic bit and the parity bit. Where y_k is the received bit for systematic and parity bits.
At the end, we calculate the extrinsic information by subtracting the decoded value from the extrinsic information from previous decoder and from the received value of the bits.

2.3. Vivado HLS

When using HLS for designing we can convert an algorithm written in HLL such as C/C++ to RTL automatically. But there are many points must deal with. First of all, the tool has limitation for example the all memories must be static and the compiler must know the memory size before the compilation. For that there is neither heap nor stack memory were used in the program [8].
Another point is how to write a program and how the tool translate the program as a hardware. For example, when we write these programming sentences in C language:

```
        if (X==Val1) X=Function(A);
        if (Y==Val2) Y=Function(B);
```

this well make the hardware of the (Function) duplicated. But if we write them like:

```
        if (X==Val1) X=Function(A);
      else if (Y==Val2) Y=Function(B);
```

the hardware well not duplicated and this well be effort on the resource utilization.
Directives or known as pragmas are another point must focus in. The hardware design can't be optimized by using HLL only. The HLS tool offer another feature for how the hardware implement-ed. There are many directives for different design's
techniques such as parallelism, pipeline, memory partitioning and many other techniques [8]. Only the directives that used in this paper well be men-tioned:

- Loop Unrolling: when this directive was used the hardware well replicated. And this well allows parallelism and effective to reduce latency but the cost is more resources well be used [8].
- Array partitioning: this directive was used to reduce the data access dispute and this done by splitting the array into small block RAM Vivado HLS provided three types of partitioning block, cyclic and complete. By using complete type, the array well split into individual elements (registers) and the fully parallel access can be done but the cost is in the resource utilization [8].

3. THE PROPOSED METHODOLOGY

The Scenario of how the MAP decoding algorithm working is presented by the following steps:

- Calculating the gamma's values and in the same time calculate alpha's values from the beginning of the trellis to the end of it.
- Calculating beta's values, decoder's output and the extrinsic information in the same time from the end of trellis to the beginning.
- Repeat the first two steps for 8 iterations.

The algorithm is implemented in C++ language using Vivado HLS tool. Vivado HLS tool supporting C, C++and System C language for designing. Also, supporting many processing tech-niques like pipeline, parallelism, memory par-titioning and many others by using directives.
In this paper unrolling directive will be used for parallelism the all for loops (the inner for loops for calculating alpha, gamma, beta and the extrinsic information. And the outer for loops for the iterations) in turbo decoder to design full parallel turbo decoder. Two designs were com-pered in this work one is the normal turbo decoder with two MAP decoders and the other is turbo decoder with one MAP decoder. Fig. 4 show how the algorithm work with one MAP decoder.
In this research, not the whole message was decoding, a smaller window size was taken. A window of 27 bits from the message to decode 9 bits and window of 108 bits to decode 36 bits. These two windows were implemented for frame size of 108, 216 and 432 to decode 36, 73 and 144 bits these numbers of frame size were chosen to fit in the two chosen windows. The window size represents the parallelism level when the Vivado

HLS unrolling directive was involved in the prog-ram loops for making them working in parallel. Block interleaver was used for the all designs. The results were calculated and comparison in latency and resource utilization were made.

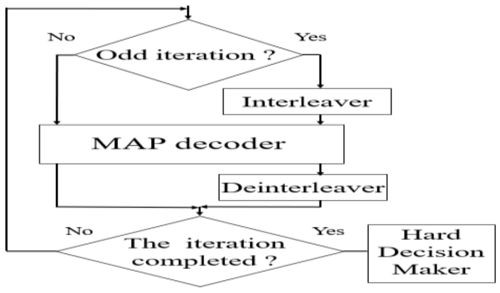

Fig. 4. Block diagram of turbo decoder with one MAP decoder.

4. RESULTS AND DISCUSSIONS

The whole programs were written in C++ language using Vivado HLS. The data type where used was double. The directives were used are dependencies and unroll for parallelism. The device where used to implement all designs was Virtex Ultra Scale+FPGA (xcvu13p-fsga2577-2-I)
 where it has 216000 SLICE, 1728000 LUT, 3456000 FF, 12288 DSP and 5476 BRAM. The targeted clock set to 10 ns.

Table 1: Results of minimum and maximum latency for turbo decoder with one MAP decoder for different designs.

Design type	Latency (clock cycles) min	Latency (clock cycles) max
Turbo decoder without parallel window 9 for 108 bits	47361	81025
Turbo decoder with parallel window 9 for 108 bits	9933	18285
Turbo decoder without parallel window 36 for 108 bits	40270	68798
Turbo decoder with parallel window 36 for 108 bits	21078	39510
Turbo decoder without parallel window 9 for 216 bits	94721	162049
Turbo decoder with parallel window 9 for 216 bits	18501	34053
Turbo decoder without parallel window 36 for 216 bits	80543	137599
Turbo decoder with parallel window 36 for 216 bits	42124	78988
Turbo decoder without parallel window 9 for 432 bits	189441	324097
Turbo decoder with parallel window 9 for 432 bits	30849	56769
Turbo decoder without parallel window 36 for 432 bits	161085	275197

The result listed in Table 1 for turbo decoder with one MAP decoder resulting the mini-mum and maximum latency with different frame size and different windows.

Table 2: Results of minimum and maximum latency for turbo decoder with two MAP decoders for different designs.

Design type	Latency (clock cycles) min	Latency (clock cycles) max
Turbo decoder without parallel window 9 for 108 bits	47841	79265
Turbo decoder with parallel window 9 for 108 bits	10534	19718
Turbo decoder without parallel window 36 for 108 bits	40486	67590
Turbo decoder with parallel window 36 for 108 bits	21558	40502
Turbo decoder without parallel window 9 for 216 bits	95681	158529
Turbo decoder with parallel window 9 for 216 bits	11614	21662
Turbo decoder without parallel window 36 for 216 bits	80975	135183
Turbo decoder with parallel window 36 for 216 bits	41784	78520
Turbo decoder without parallel window 9 for 432 bits	191361	317057
Turbo decoder with parallel window 9 for 432 bits	17953	33345
Turbo decoder without parallel window 36 for 432 bits	161949	270365
Turbo decoder with parallel window 36 for 432 bits	82114	154306

Table 3: Results of resource utilization for turbo decoder with one MAP decoder for different designs.

Design type	BRAM_ 18K	DSP48E	FF	LUT
Turbo decoder without parallel window 9 for 108 bits	24	449	29592	42238
Turbo decoder with parallel window 9 for 108 bits	0	1468	115594	142821
Turbo decoder without parallel window 36 for 108 bits	48	449	27717	42018
Turbo decoder with parallel window 36 for 108 bits	0	411	70763	53567
Turbo decoder without parallel window 9 for 216 bits	24	499	29595	42241
Turbo decoder with parallel window 9 for 216 bits	0	1468	124686	145321
Turbo decoder without parallel window 36 for 216 bits	48	499	27734	42118
Turbo decoder with parallel window 36 for 216 bits	0	411	76833	55738
Turbo decoder without parallel window 9 for 432 bits	24	499	29601	42249
Turbo decoder with parallel window 9 for 432 bits	0	1468	140090	153601
Turbo decoder without parallel window 36 for 432 bits	48	499	27741	42122
Turbo decoder with parallel window 36 for 432 bits	0	411	86157	58748

The result listed in Table 2 for turbo decod-er with two MAP decoders resulting the min and max latency for different frame size and different windows.
Where the result listed in Table 3 for turbo decoder with one MAP decoders resulting the resource utilization for different frame size and

different windows and Table 4 resulting the resource utilization for different frame size and different windows for the turbo decoder with two MAP decoders.

Table 4: Results of resource utilization for turbo decoder with two MAP decoders for different designs.

Design type	BRAM _18K	DSP48E	FF	LUT
Turbo decoder without parallel window 9 for 108 bits	24	998	57991	81599
Turbo decoder with parallel window 9 for 108 bits	0	2358	180550	226644
Turbo decoder without parallel window 36 for 108 bits	72	998	54592	81256
Turbo decoder with parallel window 36 for 108 bits	0	874	109946	107473
Turbo decoder without parallel window 9 for 216 bits	24	998	57994	81602
Turbo decoder with parallel window 9 for 216 bits	0	5502	392777	529506
Turbo decoder without parallel window 36 for 216 bits	72	998	54609	81356
Turbo decoder with parallel window 36 for 216 bits	0	874	125488	110954
Turbo decoder without parallel window 9 for 432 bits	24	998	58000	81610
Turbo decoder with parallel window 9 for 432 bits	0	8857	659167	864470
Turbo decoder without parallel window 36 for 432 bits	72	998	54616	81360
Turbo decoder with parallel window 36 for 432 bits	0	1696	244154	213636

The drawing in Fig. 6 and Fig. 7 for Table 1 and Table 2. The plotted symbols meaning listed by Table 5.
From these results we can see that when the turbo decoder with one MAP decoder was used with large window size and without using paralle-lism in the design, the latency is better from small window size and has less resource utilization only in memory utilization is higher because of the larg-er window size. And the same explanations for the turbo decoder with two MAP decoders and for maximum and minimum latency.
When we use parallelism for both turbo decoder designs with one or two MAP decoders, the minimum and maximum latency of the small window was better than the larger window size, but the resource utilization was larger than the large window size. In other hand, the larger window size takes less resources than the small window size but in latency it was not the better choice.
If we make a comparison between the turbo decoder with one MAP decoder and the turbo decoder with two MAP decoders without using parallelism the design with two MAP decoders was better in maximum latency than the design with
one MAP decoder. And if we use parallelism the turbo decoder with one MAP decoder was the best for the two designs in maximum latency. And it was utilized less resources than the turbo decoder with two MAP decoders and for the two designs with or without using parallelism.

Table 5: Fig. 6 and Fig. 7 Chart's symbols meaning.

W9 for min	Turbo decoder without parallel window 9 for min latency
W9P for min	Turbo decoder with parallel window 9 for min latency
W36 for min	Turbo decoder without parallel window 36 for min latency
W36P for min	Turbo decoder with parallel window 36 for min latency
W9 for max	Turbo decoder without parallel window 9 for max latency
W9P for max	Turbo decoder with parallel window 9 for max latency
W36 for max	Turbo decoder without parallel window 36 for max latency
W36P for max	Turbo decoder with parallel window 36 for max latency

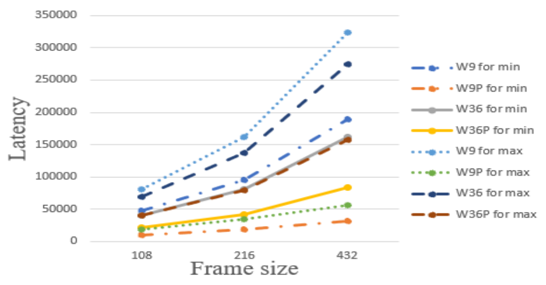

Fig. 6 Latency plot for turbo decoder with one decoder.

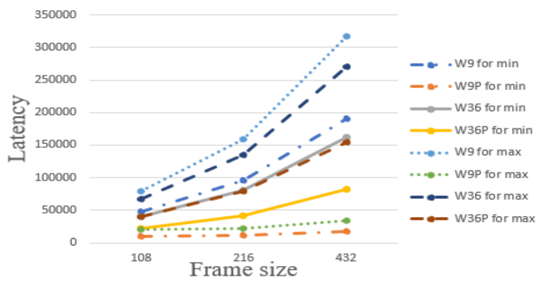

Fig. 7 Latency plot for turbo decoder with two decoders.

5. CONCLUSION

In this paper we explained that we can use HLS to make an architecture design by C++ language and make the (RTL) automatically using Vivado HLS and make several designs with many processes like parallelism, and we conclude that the design of turbo decoder with one MAP decoder is better than the turbo decoder with two decoders spatially when we went to make the area of the

designs smaller by utilize less resources. And when we went to design low maximum latency turbo
decoder in fully parallel design, the turbo decoder with one MAP decoder produce better results.

REFERENCES

[1] H. Luo *et al*, "Low Latency Parallel Turbo Decoding Implementation for Future Terrestrial Broadcasting Systems," in *IEEE Transactions on Broadcasting*, vol. 64, no. 1, pp. 96-104, 2018.

[2] S. Shao et al., "Survey of Turbo, LDPC, and Polar Decoder ASIC Implementations," in IEEE Communications Surveys & Tutorials, vol. 21, no. 3, pp. 2309-2333, 2019.

[3] F. Shaheen, M. F. U. Butt, S. Agha, S. X. Ng and R. G. Maunder, "Performance Analysis of High Throughput MAP Decoder for Turbo Codes and Self Concatenated Convolutional Codes," in *IEEE Access*, vol. 7, pp. 138079-138093, 2019.

[4] J. Andrade *et al.*, "Design Space Exploration of LDPC Decoders Using High-Level Synthesis," in *IEEE Access*, vol. 5, pp. 14600-14615, 2017.

[5] B. E. Conn, "Exploring High Level Synthesis to Improve the Design of Turbo Code Error Correction in a Software Defined Radio Context," Master's thesis, pp. 109, 2018.

[6] S. Lahti, P. Sjövall, J. Vanne and T. D. Hämäläinen, "Are We There Yet? A Study on the State of High-Level Synthesis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898-911, 2019.

[7] W. Stirk and J. Goeders, "Implementation and Design Space Exploration of a Turbo Decoder in High-Level Synthesis, "2019 International Conferenceon ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, pp. 1-5, 2019.

[8] Xilinx, "Vivado HLS References," vol. 901. pp. 1–120, 2018.

[9] M. F. Brejza, R. G. Maunder,B. M. Al-Hashimi and L. Hanzo,"A High-Throughput FPGA Architecture for Joint Source and Channel Decoding," in *IEEE Access*, vol. 5, pp. 2921-2944, 2017.

[10] J. Andrade et al., "Design Space Exploration of LDPC Decoders Using High-Level Synthesis," in IEEE Access, vol. 5, pp. 14600-14615, 2017.

[11] A. Badr, A. Khisti, W. Tan and J. Apostolopoulos, "Perfecting Protection for Interactive Multimedia: A survey of forward error correction for low-delay interactive applications," in *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 95-113, March 2017.

[12] G. Tzimpragos, C. Kachris, I. B. Djordjevic, M. Cvijetic, D. Soudris and I. Tomkos, "A Survey on FEC Codes for 100 G and Beyond Optical Networks," in *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 209-221, Firstquarter 2016.

[13] M. D. S. P. Design, "Vivado HLS Lab Tutorial," *Fpga*, vol. 986. pp. 1–100, 2014.

[14] M. F. U. Butt, S. X. Ng and L. Hanzo, "Self-Concatenated Code Design and its Application in Power-Efficient Cooperative Communications," in *IEEE Communications Surveys & Tutorials*, vol. 14, no. 3, pp. 858-883, Third Quarter 2012.

[15] Chiueh, Tzi-Dar, et al. *Baseband receiver design for wireless MIMO-OFDM communications*. Singapore: Wiley, 2012.

[16] Yahya T. Qassim, Dhafir A. Alneema, "FPGA Based Implementation of Convolutional Encoder- Viterbi Decoder Using Multiple Booting Technique". *AL-Rafdain Engineering Journal (AREJ)*, Vol.18, No.6, pp. 70-80, 2010.

تحليل تصميم فك الترميز توربو المرتكزعلى فك الترميز تكبير احتمال خلفي **(MAP)** وحيد باستخدام أداة تركيب عالية المستوى **(HLS)**

عامر طلال علي
aamertali@yahoo.com

ظافر عبد الفتاح عبد القادر
dhafir.abdulfattah@uomosul.edu.iq

جامعة الموصل - كلية الهندسة - قسم هندسة الحاسوب

الملخص

إن أداة التركيب عالية المستوى (HLS) لا تسهل عملية التصميم وتسريع النماذج فقط وانما تسمح للمصممين استكشاف عدد كبير من تقنيات التصميم مثل الموازاة، خطوط الانابيب، تقسيم الذاكرة والكثير من التقنيات الأخرى فكالترميزتوربو المرتكز على خوارزمية تكبير الاحتمال الخلفي (MAP) قد صممت في هذا العمل بإستخدام برنامج Vivado HLS. تم في هذا البحث تنفيذ فك الترميز توربو الاعتيادي المكون من اثنين من فك ترميز تكبير الاحتمال الخلفي(MAP) واقتراح تصميم لفك الترميز توربو جديد يتمثل باستخدام فك ترميز تكبير الاحتمال الخلفي (MAP)واحد وقد تم تنفيذ التصميمين باستخدام الموازاة وبدونها في كلا التصميمين وبإستخدام تقنية النافذة وباستخدام برنامج Vivado HLS وان هذا العمل لم يتم تنفيذه سابقا. هذه التصاميم نفذت في هذا العمل على نماذج ذات احجام مختلفة. تم عمل مقارنة في زمن الاستجابة واستغلال المصادر وكيفية عمل مقايضة بين هاذين العاملين وذلك للوصول الى التصميم المناسب الذي نحتاجه.

الكلمات الدالة:

أداة تركيب عالية المستوى، فك الترميز توربو، زمن الاستجابة، استغلال المصادر.