THE REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

DESIGN AND IMPLEMENTATION OF HIGH THROUGHPUT BIDIRECTIONAL FANO DECODING

M.S. Thesis

AHMET KAKACAK

İSTANBUL, 2012

THE REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
ELECTRICAL & ELECTRONICS ENGINEERING

DESIGN AND IMPLEMENTATION OF HIGH THROUGHPUT BIDIRECTIONAL FANO DECODING

M.S. Thesis

AHMET KAKACAK

Advisor: Prof. TAŞKIN KOÇAK

İSTANBUL, 2012

THE REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
ELECTRICAL & ELECTRONICS ENGINEERING

Name of the thesis: Design and Implementation of High Throughput Bidirectional Fano Decoding

Name/Last Name of the Student: Ahmet KAKACAK

Date of the Defense of Thesis: September 5, 2012

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc. Prof. Tunç BOZBURA
Director

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Assoc. Prof. Ufuk TÜRELİ
Program Coordinator

Examining Comittee Members	Signature
Prof. Taşkın Koçak	----------
Assoc. Prof. Ufuk TÜRELİ	----------
Assist. Prof. Çağrı GÜNGÖR	----------

ABSTRACT

DESIGN AND IMPLEMENTATION OF HIGH THROUGHPUT BIDIRECTIONAL FANO DECODING

Ahmet Kakacak

Electrical & Electronics Engineering

Thesis Supervisor: Prof. Taşkın Koçak

September 2012, 50 pages

This study deals with hardware implementation of the bidirectional Fano decoding. Sequential decoding techniques such as Fano provide lower hardware complexity when compared with the Viterbi decoding. These techniques are adaptive to different SNR values and they have variable computational delay, so the throughput is really high at high SNR values. But the throughput drops significantly at low SNR values. Bidirectional decoding technique can be applied to increase the throughput and reduce the computational variability. Two decoders work inversely in parallel and they merge at any position in a codeword. The merger depends on some circumstances. States of the two decoders must be equivalent at the merger position. If number of checked states is chosen higher than one, neighbor states also must be equivalent.

Software implementations show that the bidirectional Fano decoding reduces computational effort, but it must be proved in hardware too. In software, only number of iterations can be measured and also conventional processors do not let high throughput. But in hardware, we can achieve high throughput. This study bridges a gap in this topic. Since FPGAs are widely used in telecommunications industry, implementation results in FPGAs are very important. Here is a chance to compare hardware implementation results of the unidirectional Fano decoding and bidirectional Fano decoding. And also comparison of the Fano decoding and the Viterbi decoding is available in terms of clock frequency and FPGA resources consumed.

Keywords: Fano Decoder, Bidirectional Fano Decoding, Sequential Decoding

ÖZET

YÜKSEK HIZLI ÇİFT YÖNLÜ FANO KOD ÇÖZÜMÜNÜN
TASARIM VE GERÇEKLEMESİ

Ahmet Kakacak

Elektrik-Elektronik Mühendisliği

Tez Danışmanı: Prof. Taşkın Koçak

Eylül 2012, 50 sayfa

Bu çalışma çift yönlü Fano kod çözümünün donanımsal gerçeklemesini kapsamaktadır. Fano gibi ardışıl kod çözümü yöntemleri Viterbi'ye kıyasla daha düşük donanım karmaşıklığı sağlamaktadır. Bu yöntemler farklı işaret/gürültü oranlarına uyarlanabilir ve değişken hesaplama süresine sahiptir, bu nedenle yüksek işaret/gürültü oranlarında yapılan iş oldukça yüksektir. Fakat düşük işaret/gürültü oranlarında yapılan iş bir hayli düşer. Yapılan işi artırmak ve hesaplama süresindeki değişkenliği azaltmak için çift yönlü kod çözümü yöntemi uygulanabilir. İki kod çözücü ters yönlerde paralel olarak çalışır ve kod dizisinin herhangi bir yerinde birleşir. Bu birleşme iki kod çözücünün durumlarının aynı noktada eşit olmasına bağlıdır. Eğer kontrol edilen durum sayısı birden fazlaysa komşu durumlar da eşit olmalıdır.

Yazılım ortamında yapılan çalışmalar çift yönlü Fano kod çözümünün, işlem yükünü azalttığını göstermektedir. Fakat söz konusu hipotez donanımsal olarak da kanıtlanmalıdır. Yazılımda sadece iterasyon sayısı gözlemlenebilir, ayrıca geleneksel işlemciler yapılan işin yüksek olmasına izin vermez. Fakat donanımda yapılan iş çok daha yüksek olabilir. Bu çalışma bu konudaki boşluğu dolduracaktır. Haberleşme sektöründe FPGA'ler yaygınca kullanıldığı için FPGA üzerindeki gerçekleme sonuçları oldukça önemli. Burada tek yönlü ve çift yönlü Fano kod çözümünün donanımsal verilerini karşılaştırma olanağı bulunacaktır. Ayrıca saat frekansı ve tüketilen FPGA kaynakları açısından Fano ve Viterbi kod çözümleri de karşılaştırılabilecektir.

Anahtar Kelimeler: Fano Kod Çözücü, Çift Yönlü Fano Kod Çözümü, Ardışıl Kod Çözümü

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC	: Application-Specific Integrated Circuit
AWGN	: Additive White Gaussian Noise
BER	: Bit Error Rate
BFA	: Bidirectional Fano Algorithm
BMSA	: Bidirectional Multiple Stack Algorithm
BPSK	: Binary Phase-Shift Keying
BRAM	: Block RAM
CRC	: Cyclic Redundancy Check
DCM	: Digital Clock Manager
FEC	: Forward Error Correction
FF	: Flip-Flop
FPGA	: Field-Programmable Gate Array
GSM	: Global System for Mobile Communications
HDL	: Hardware Description Language
IP	: Intellectual Property
I/O	: Input/Output
LAN	: Local Area Network
LUT	: Look-Up Table
MSA	: Multiple Stack Algorithm
PLL	: Phase Locked Loop
RAM	: Random-Access Memory
RTL	: Register Transfer Level
SNR	: Signal-to-Noise Ratio
UCF	: User Constraint File
UFA	: Unidirectional Fano Algorithm

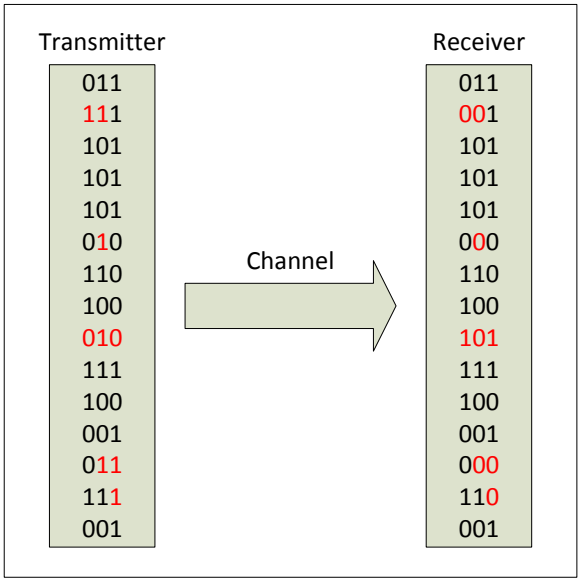
1. INTRODUCTION

1.1 MOTIVATION

In telecommunication and computer networking, errors may be introduced during data transmission. These errors affect even workaday life. For example, sound quality of a phone call considerably depends on the transmission.

Errors occur mostly by reason of channel noise. If digital data is the point at issue, some bits of the data may be flipped until reaching a receiver. Figure 1.1 shows an example set of transmitted and received data. The bits that had been flipped during transmission are shown in red color. These received data are not interpreted yet.

Figure 1.1: Transmitted and received data



Error detection and correction techniques can be used to eliminate errors and reconstruct or retransmit the original data. The repetition code below is a basic example of error-correcting codes. All bits are transmitted three times and interpreted as follows.

Table 1.1: Repetition code

Received	000	001	010	011	100	101	110	111
Interpreted	0	0	0	1	0	1	1	1

1.2 SCOPE OF THE THESIS

This thesis aims to design a decoder which provides high throughput but consumes little hardware resources. The decoder uses the bidirectional Fano algorithm and chapter 2 explains why this algorithm is chosen.

The Fano algorithm has variable throughput at output, so it can be sped up by using bidirectional decoding manner. One of the most important points of this thesis is proving that bidirectional Fano decoder is efficient in terms of throughput/area. Since bidirectional fano decoder is expected to consume two times more hardware resources than unidirectional fano decoder, it has to provide much more throughput gain.

Alternatively the Viterbi algorithm is widely (see chapter 2.4) used in convolutional decoders, then the performance of the Fano algorithm becomes very important against the Viterbi algorithm. One of the aims of this thesis is showing that the bidirectional Fano decoder can compete with the Viterbi decoder especially at high signal-to-noise ratio (*SNR*).

1.3 TARGET PLATFORM

Implementation of the Fano algorithm is in Verilog which is a hardware description language (*HDL*). Target devices are field-programmable gate arrays (*FPGAs*) that are widely used in communication industry. According to Xilinx's fiscal 2012 results, 41% of FPGAs are used in this industry.

Since the FPGA vendor Xilinx Inc. has a Viterbi decoder IP, Xilinx FPGAs are very suitable as target platform. This decoder IP is acceptable as a reference design for comparison of the Viterbi algorithm and the Fano algorithm. Because its constraint length is equivalent to the Fano decoder designed in this thesis. The constraint length equals to seven.

The unidirectional and bidirectional Fano decoders were simulated in ModelSim. Then they were synthesized and analyzed by Xilinx ISE Design Suite. Timing results and hardware resources discussed in this thesis are valid for only the target FPGA families (e.g. Spartan-6). But the designs are not dependent on this FPGAs and they can be synthesized for other FPGAs or ASICs too.

Specifications of the Viterbi decoder were quoted from the document which is labeled as *DS247* and published by Xilinx Inc. The specifications in this document belongs to Viterbi Decoder v7.0 (Xilinx Inc. 2011) and they are used as reference for the Viterbi decoder.

1.4 CONTRIBUTIONS OF THE THESIS

The contributions of this thesis are as follows:

i) Hardware complexity of the bidirectional Fano decoder is revealed and a faster decoder is designed with respect to the unidirectional Fano decoder.

ii) It is proved that the Fano decoder can work at least as fast as the Viterbi decoder at high SNR values per FPGA resources consumed.

1.5 OUTLINE OF THE THESIS

In chapter 2, historical information about a few error correction techniques is given. This chapter is ordered by beginning from general topics.

In chapter 3, the main structure of the Fano algorithm is explained.

In chapter 4, it is explained that how the Fano decoder is designed. Then some information about the simulation and synthesis environments is given.

In chapter 5, simulation and synthesis results are given. In chapter 6, the results are discussed and some information is given about possible future work.

2. LITERATURE REVIEW

2.1 REVIEW OF FORWARD ERROR CORRECTION

In 1940s Richard Wesley Hamming began to work on error-correcting codes and in 1950 he invented the Hamming code which is the first error-correcting code suitable for forward error correction (*FEC*).

Error-correcting codes could be used for error detection, but in many applications such as a TV broadcast, retransmission of erroneous data is not possible. So a receiver does not only detect errors, but also tries to reconstruct error-free data. This reconstruction process is called forward error correction.

Since error-correcting codes are generally divided into two types as block codes and convolutional codes, FEC codes also can be investigated in these types. The repetition code given in Table 1.1 is an example of block codes. Convolutional codes will be elaborated in the next chapters.

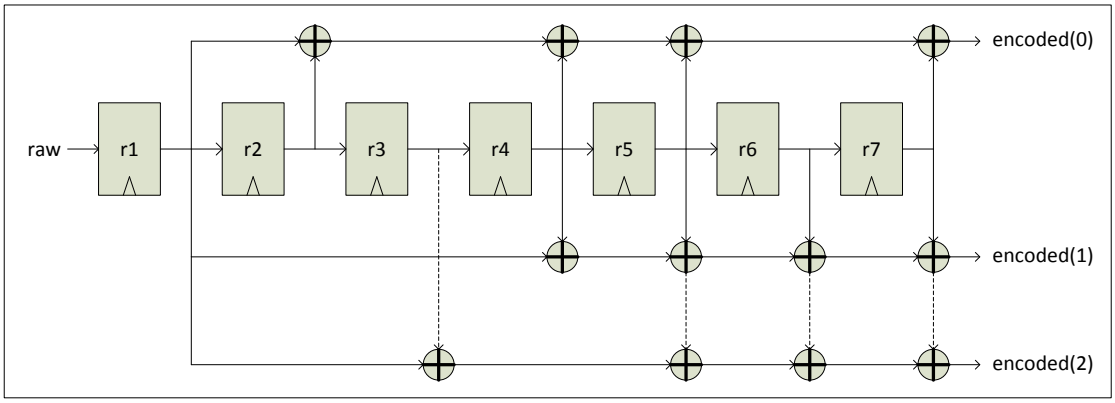
Decoding techniques can be divided into two with respect to the decision types, hard-decision decoding and soft-decision decoding. Input symbols of a hard-decision decoder consist of exact zeros and ones. But input symbols of a soft-decision decoder consist of probabilities. For example, let the soft width be three. "000" may represent a strong zero and "010" may represent a weak zero.

2.2 REVIEW OF CONVOLUTIONAL CODING

Convolutional coding is a major type of FEC techniques (Zhou and others 2006) and it was introduced by Peter Elias in 1955. This technique is widely used in mobile communication and satellite communication (Mohammad and others 2008).

In convolutional coding, each m-bit symbol is encoded in n bits, then m/n is called code rate ($m \leq n$). Number of last symbols which are used to generate a new (encoded) symbol is called constraint length. Figure 2.1 shows a convolutional encoder whose code rate (R) is 1/3 and constraint length (K) is seven.

Figure 2.1: Convolutional encoder



The generator polynomials of the encoder are as follows:

$G_o = r1 + r2 + r4 + r5 + r7$

$G_1 = r1 + r4 + r5 + r6 + r7$

$G_2 = r1 + r3 + r5 + r6 + r7$

where + is exclusive-OR (XOR) operator.

2.3 REVIEW OF SEQUENTIAL DECODING

Different techniques are used to decode convolutional codes, sequential decoding is one of them. Sequential decoders make computations for only one path at a symbol interval, but Viterbi decoder makes computations for all possible paths (Long and Bush 1989). This reason makes sequential decoders less complex than Viterbi decoder in terms of computational effort.

Complexity of sequential decoders does not depend on constraint length as much as Viterbi decoder. In equivalent complexity a sequential decoder may have longer constraint length and it can achieve a lower (better) bit error rate (*BER*).

Sequential decoding was introduced by John McReynolds Wozencraft in 1957 and there are two mostly known sequential decoding algorithms: The Stack algorithm and the Fano algorithm.

2.3.1 Review of Bidirectional Sequential Decoding

In 1967, George David Forney proposed that convolutional codes can be decoded from the end if the code ends in a known state. Then forward decoder (*FD*), which starts from the beginning of the code, and backward decoder (*BD*), which starts from the end of the code, may process simultaneously. FD and BD merge at a common state.

Bidirectional sequential decoding reduces computational variability when compared with unidirectional sequential decoding as proposed by Kaiping Li and Samir Kallel (1991). The bidirectional multiple Stack algorithm (*BMSA*) can give better results than the multiple Stack algorithm (*MSA*) in terms of computational effort and bit error rate (Li and Kallel 1999).

2.4 REVIEW OF VITERBI ALGORITHM

The Viterbi algorithm is a maximum likelihood (*ML*) convolutional decoding algorithm which was proposed by Andrew James Viterbi (1967). It has different modified versions such as the list Viterbi algorithm and the M-algorithm. This algorithm is widely used in GSM, satellite and wireless local area networks (*LANs*).

While constraint lengths are equivalent, the Viterbi algorithm provides lower BER than sequential decoding algorithms. But, if constraint length increases, hardware complexity increases exponentially. So Viterbi algorithm becomes too complex especially for constraint lengths greater than eight (Ran and others 2009). Then sequential decoding algorithms have an advantage as mentioned in chapter 2.3.

2.4.1 Review of List Viterbi Algorithm

If the convolutional code is concatenated with an outer block code such as cyclic redundancy check (*CRC*) code, then the Viterbi algorithm is not the ML decoder anymore. The list Viterbi algorithm can be used in this case. This algorithm finds the ordered list of the L sequences and then CRC decoder selects an appropriate path from the list (Chen and Sundberg 2000).

2.4.2 Review of M-Algorithm

The M-algorithm is a simplified version of the Viterbi algorithm and its complexity does not depend on the constraint length. The M-algorithm works the same as the Viterbi algorithm, but it keeps only *M* paths with the largest metrics. However this property reduces complexity, there is a possibility to lose correct path. If *M* is a small number, the possibility is high. After the correct path is lost, the frame may be decoded with a dramatically high BER. Adaptive versions of the M-algorithm are created to avoid losing the correct path. The algorithm mostly uses a small value of *M* and big values are used only if necessary (Zadeh and Soleymani 2005).

2.5 REVIEW OF FANO ALGORITHM

The Fano Algorithm is a sequential decoding algorithm which was introduced by Robert Mario Fano in 1963. At great constraint lengths its implementation can be simpler than the Viterbi algorithm as mentioned in chapter 2.4; furthermore, since memory requirement of the Stack algorithm is higher, the Fano algorithm becomes very suitable for hardware implementation (Long and Bush 1989).

By reason of computational variability, Fano decoder requires a large input buffer. This is one of the most important problems of the Fano algorithm. The buffer can overflow when the decoder slows down. But some control mechanisms were introduced such as increasing the delta (Δ) parameter of the Fano. If the delta is high, bit error rate increases but also the decoder runs significantly faster. So speed of the decoder can be controlled to be able to avoid buffer overflow as proposed by Pan and Ortega (2001).

Implementation in a Xilinx Virtex-4 FPGA shows that the Fano decoder consumes 12.3% of the equivalent resources that the Viterbi decoder consumes (Benaissa and Yiqun 2007). Maximum throughput of the Fano decoder is 75 Mbps and maximum throughput of the Viterbi decoder is 35 Mbps.

2.5.1 Review of Bidirectional Fano Algorithm

Bidirectional sequential decoding technique can be applied to the Fano algorithm as well. The analysis made by Ran and others (2009) shows that the bidirectional Fano algorithm (*BFA*) provides throughput improvement up to 300% with respect to the

unidirectional Fano algorithm (*UFA*). Since bidirectional sequential decoding reduces computational variability as mentioned in chapter 2.3.1, the bidirectional Fano decoder is very suitable for parallelism with respect to the unidirectional Fano decoder (Ran and others 2009).

2.6 REVIEW OF STACK ALGORITHM

The Stack algorithm was introduced by Kamil Sh. Zigangirov (1966) and independently by Frederick Jelinek (1969). It is also known as the *JZ algorithm*. The Stack algorithm stores the paths in stack in order of descending f-function values (Han and Chen 2002). If the number of paths is high during the search, the stack can overflow. One solution to avoid the overflow problem is discarding the path with the smallest f-function value (Lin and Costello 1983).

Bidirectional sequential decoding technique can be applied to the Stack algorithm. The bidirectional Stack algorithm uses two stacks. One stack is for FD and the other one is for BD. Studies show that the bidirectional Stack algorithm reduces decoding effort when compared with the conventional Stack algorithm (Senk and Radivojac 1997).

To improve performance of the Stack algorithm, the multiple Stack algorithm was developed. The multiple Stack decoder has a central processor and a number of finite-size stacks (Chevillat and Costello 1997). This algorithm reduces decoding delay, but it needs more memory. The multiple Stack algorithm also can work in bidirectional decoding manner. This new algorithm is called the bidirectional multiple Stack algorithm (Li and Kallel 1999).

3. FANO ALGORITHM

3.1 UNIDIRECTIONAL FANO ALGORITHM

The Fano algorithm is a sequential decoding algorithm that has variable decoding delay. It can move forward or backward in the codeword. Each iteration determines the next direction by comparison of the path metric and the threshold (*T*). After each iteration, the threshold is adjusted by adding or subtracting the delta.

Since the encoder starts from state zero, the Fano decoder does the same. If the encoder aims to return back to state zero, it adds zeros to the end of the codeword and it generates (encodes) tail symbols. Then the decoder decodes the tail bits too and again it tries to return to state zero. While the decoding is unidirectional, tail bits are not necessary but they improve BER performance.

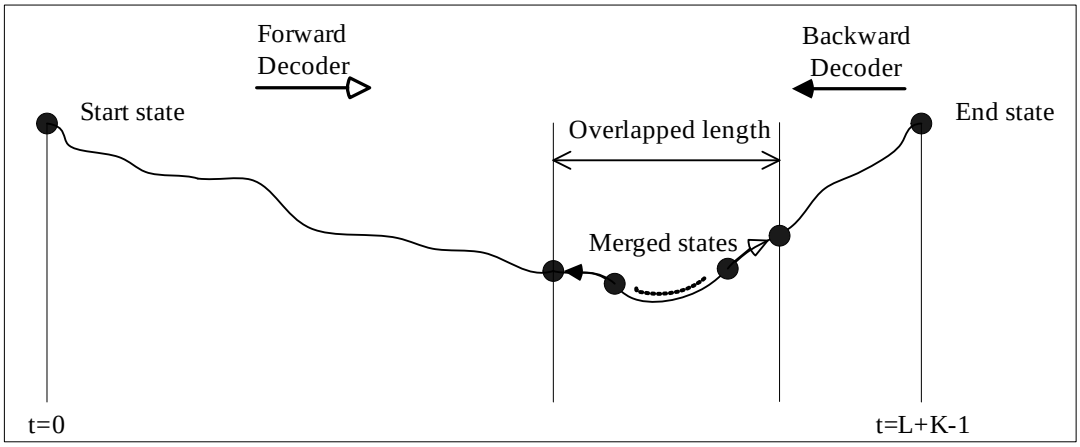
Number of tail bits is *K-1*,then length of the whole codeword equals to *L+K-1*, where *L* is the length of the original codeword and *K* is the constraint length. For example; if *L* equals to 200 and *K* equals to seven, the encoder generates 206 symbols and the decoder decodes 206 bits. Of course, the last six bits equal to zero.

However Figure 3.2 shows the flowchart of the bidirectional Fano algorithm; if the blue shaded box and the parts including the gap are removed, the figure shows the unidirectional Fano algorithm.

3.2 BIDIRECTIONAL FANO ALGORITHM

The Fano algorithm can decode a codeword bidirectionally as mentioned in chapter 2.3.1 and chapter 2.5. Both forward decoder and backward decoder start from state zero. Then they merge at the same position within the codeword. FD and BD must have the same state to be able to merge. However they can merge by checking only one state, they can check more consecutive states too. If number of checked states is increased, BER performance is improved.

Figure 3.1: Merger of forward decoder and backward decoder



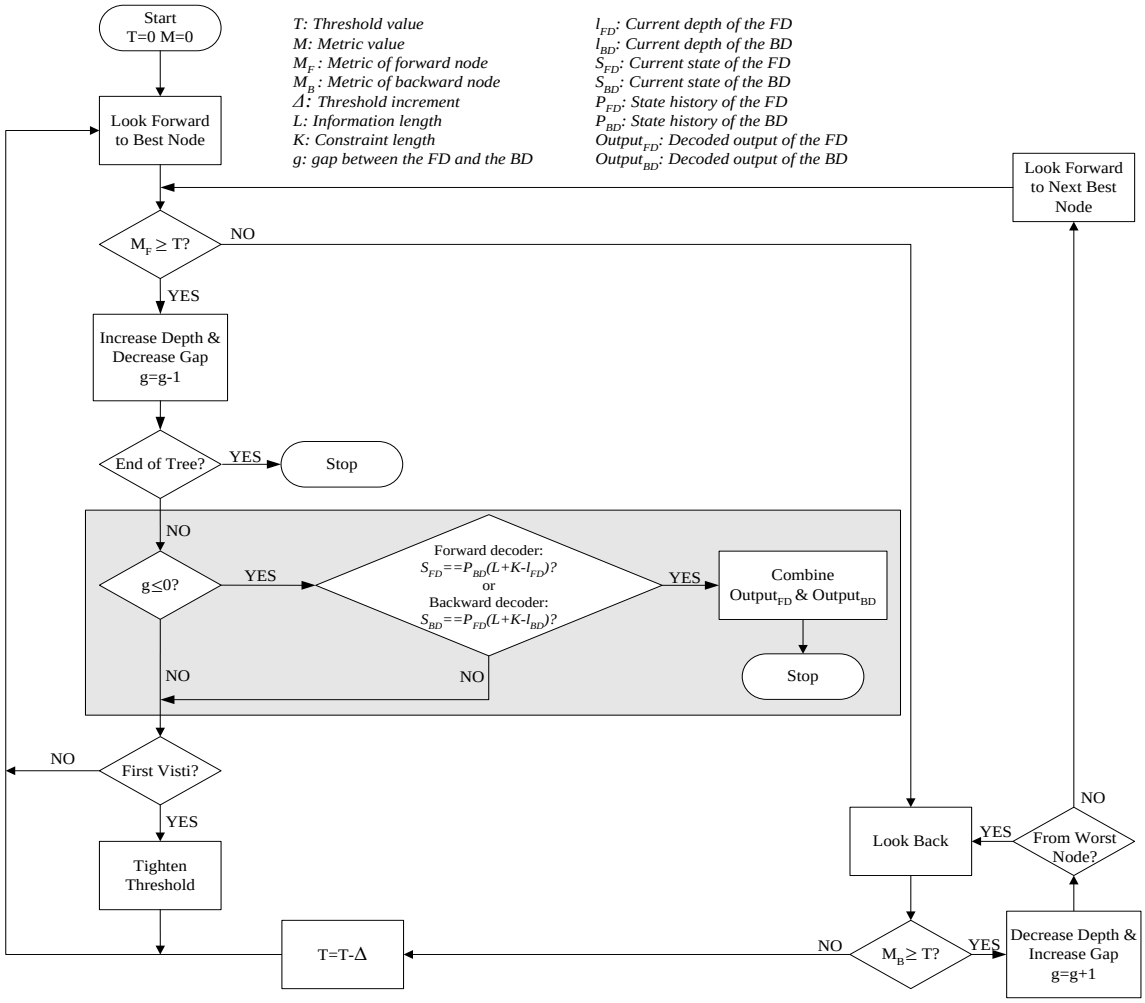
Source: Ran and others, 2011.

The difference between the current depth of the FD and that of the BD is called gap. If the decoders overlap and the gap is greater or equal to the target number of merged states, the past states are checked for the merger. The position that the FD and the BD merges called merging depth. Outputs of the decoders till the merging depth are combined into a single output codeword.

The merging process can be seen in Figure 3.1. FD and BD start decoding from inverse points of the codeword. The point is zero for FD and L+K-1 for BD. Vertical location of the line shows states. If overlapped length is greater or equal to merged states, the latest states are compared. If the states are equal by beginning from the position of FD, the decoder merges at current depth of FD. If the states are equal by beginning from the position of BD, the decoder merges at current depth of BD. If the states are not equivalent at some positions, FD and BD continue decoding.

Flowchart of the bidirectional Fano algorithm is shown in Figure 3.2.

Figure 3.2: Flowchart of bidirectional Fano algorithm



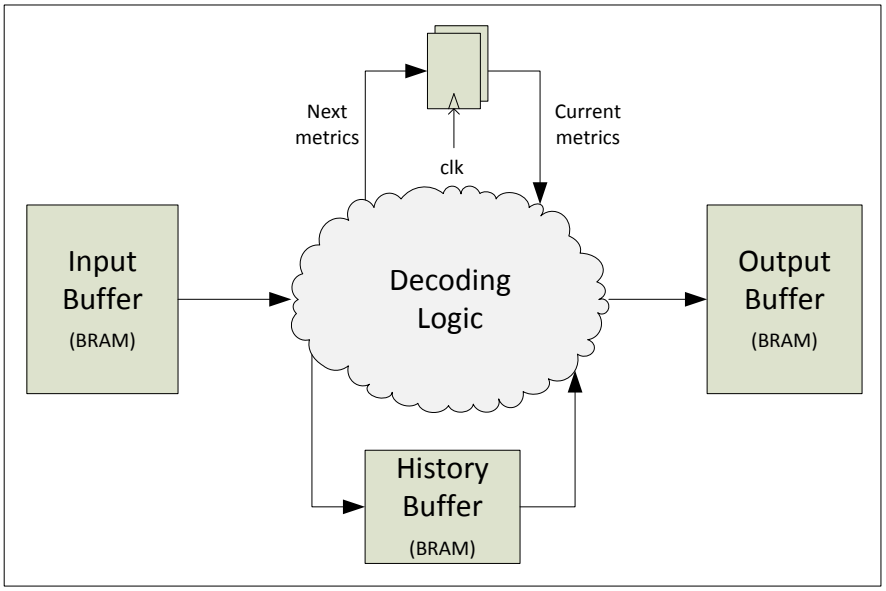
Source: Ran and others, 2011.

4. DESIGN AND IMPLEMENTATION

4.1 HARDWARE ARCHITECTURE OF FANO DECODER

The unidirectional fano decoder designed has an input buffer and an output buffer. Since the length of the original codeword equals to 200 and the constraint length equals to seven, depth of both buffers is 206 (see chapter 3.1). Data width of the input buffer is three, because the code rate is 1/3 and each input symbol of the decoder is in three bits. Data width of the output buffer equals to one.

Figure 4.1: Main structure of Fano decoder



The decoder stores some historical data in another buffer. The data in this history buffer is used when the decoder moves backward. Depth of the buffer is 206 as the input buffer and output buffer. Width of the data stored per symbol is 81, but different signals use separate buffers. The signals are shown below:

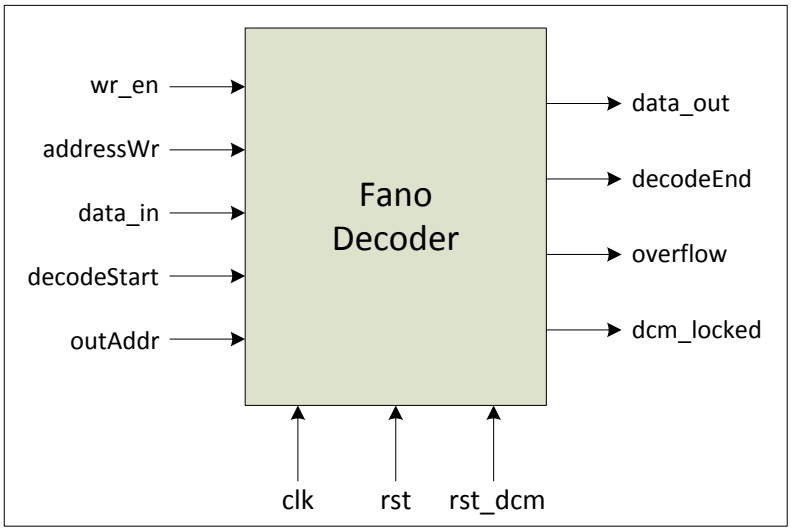
- Path metric history (10-bit)
- State history (6-bit)
- Flag history (1-bit)
- Visit record (64-bit)

If the decoder moves forward, it uses the data which is stored in flip-flops in previous clock cycle. Other parts of the decoder are mostly combinatorial.

4.1.1 Input/Output Signals

Input/output signals of the decoder designed are not based on any standard architecture. These signals were determined according to the simulation environment. Figure 4.2 shows what the I/O signals are. These signals are the same in the unidirectional and bidirectional Fano decoders.

Figure 4.2: Input/output signals



Clk is the input clock of the decoder. *Rst_dcm* resets the internal clock source which can be a phase-locked loop (*PLL*) or digital clock manager (*DCM*). Rst signal resets other parts of the decoder. *Dcm_locked* rises when the internal clock source is locked. *Rst* must be released after this signal rises.

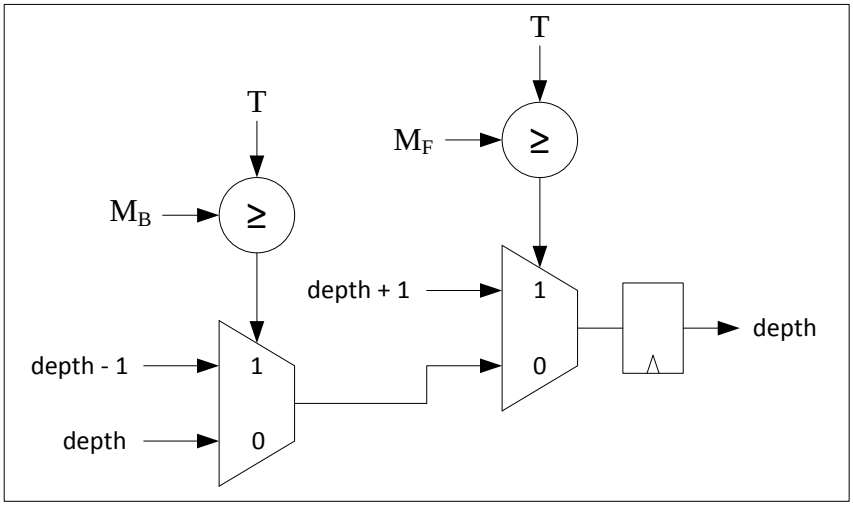
Encoded data is received via *data_in* which is a 3-bit signal. The data is written to the input buffer. *Wr_en* signal is active while the data is being received and *addressWr* represents the address which the data is written. Depth of the input buffer equals to 206 as mentioned in chapter 4.1, so *addressWr* is a 8-bit signal. After all input data is written to the input buffer, *decodeStart* is triggered as a one-cycle pulse and the decoder starts decoding. After a codeword is decoded or the overflow limit (see chapter 4.1.5) is exceeded, *decodeEnd* signal notifies that the decoder had finished decoding. If the overflow limit is exceeded, *overflow* signal is activated as a one-cycle pulse.

After the decoding process is finished, an external module can read the decoded data from the output buffer. *OutAddr* is a 8-bit address signal driven by the external module. *Data_out* which is a 1-bit signal gives the output data with one clock cycle latency.

4.1.2 Decision of Decoding Direction

Since each clock cycle represents a decoding iteration, the decoder makes direction decisions in all clock cycles. After comparison of the path metrics with the threshold, there are three possibilities. The decoder moves forward or backward or it stays at the same position. Figure 4.3 shows the decision circuit.

Figure 4.3: Circuit of direction decision



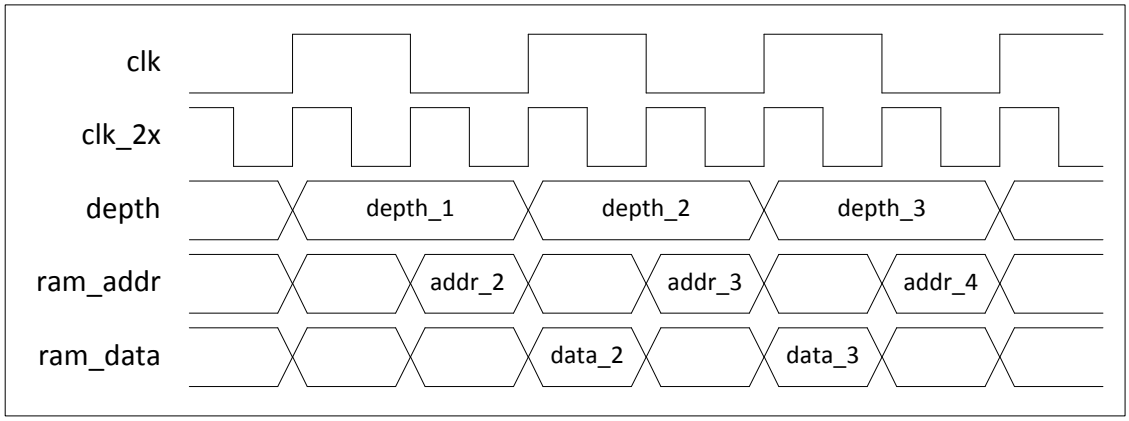
The path metric forward (M_F) which is calculated in previous clock cycle has the most priority. If it is greater or equal to the threshold, the decoder moves forward. If not, the path metric back (M_B) of the previous depth is used. If M_B is greater or equal to the threshold, the decoder moves backward; else it keeps the position.

4.1.3 Driving Buffers

The Fano algorithm has variable decoding delay, so address generation of RAMs (buffers) depends on the decoding direction. After decision of the direction, the data must be ready in the output port of the RAMs in the next clock cycle. By this reason, signal assignments to the inputs of the input buffer and history buffer can not be made via flip-flops that are using the same clock. A faster clock may be used or the assignments are made without any flip-flop.

However the input and history buffers have such a problem, the output buffer does not. No data is read from the output buffer until the end of decoding a codeword, so writing delay of decoded data is not important.

Figure 4.4: Waveform of RAM signals for 1st architecture

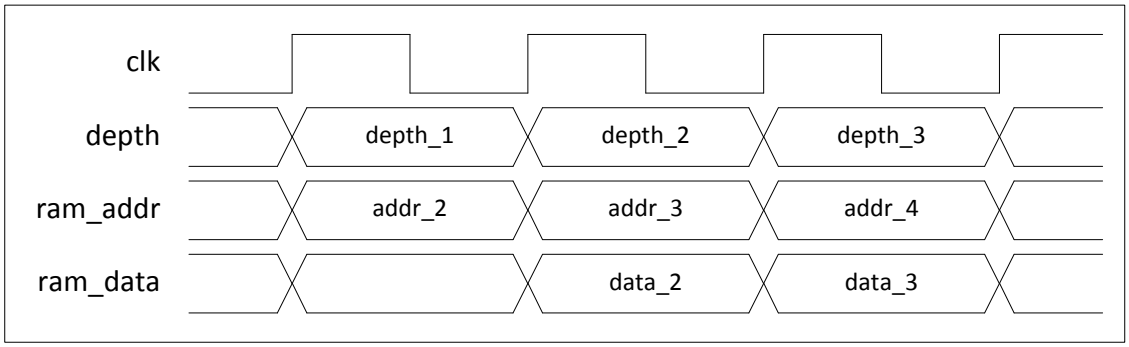


Logic delays are ignored.

The waveform in Figure 4.4 shows how a RAM is driven by a clock (*clk_2x*) which is two times faster than the system clock (*clk*). *Clk* and *clk_2x* are synchronous clocks. In the first half of the clk cycle, direction decision is made. In the second half, control signals of the RAM are assigned. Then the data is read in the next clk cycle. *Clk_2x* may be generated by a PLL or DCM which exist as hard IPs in the FPGAs. This architecture will be called *the first architecture* in the next chapters.

Since direction decision is made near the end of the logic, critical path of the logic can not be divided into two parts equally. Most part of the logic is forced to work with *clk_2x* and frequency of *clk* is limited too much. So buffers may be driven by the signals which are assigned without flip-flops. This approach may enable faster clocks and a PLL or DCM is not required. In this architecture, critical path starts from the input buffer and ends in the output buffer.

Figure 4.5: Waveform of RAM signals for 2nd architecture



Logic delays are ignored.

The waveform in Figure 4.5 shows how a RAM is driven by *clk*. The address of the RAM is assigned as soon as direction decision is made without any flip-flops. Then the data is ready in the next clock cycle. This architecture will be called *the second architecture* in the next chapters.

4.1.4 Visit Record

The visit record shows previously what states are used for each symbol. Since there are 64 possible states, 64-bit data are kept in a buffer. The buffer should be full of zeros before starting to decode a codeword. The unidirectional and bidirectional Fano decoders designed simulates only one codeword in a simulation. So it uses a RAM which is initialized with zeros while FPGA is being configured.

After 64-bit data is read from the buffer, one bit is selected by the current state. The selected bit shows whether the current state is being used for the first time. If the bit equals to zero, it means it is the first visit, then the threshold is updated. If the bit is one, it is not the first visit and the threshold keeps its previous value.

After the bit which represents the visit record of the current state is toggled to one, 64-bit data is written to the buffer back. If it is not the first visit, 64-bit data written is equivalent to the data read.

4.1.5 Overflow Limit

Since the Fano algorithm has a variable decoding delay, there must be an overflow limit to decode a codeword. In hardware, the limit is set in terms of clock cycle. As soon as the decoding starts, also a counter starts to count clock cycles. The decoder stops

decoding the current codeword when the counter exceeds the limit. And then the decoder behaves as if it decoded the entire codeword.

The overflow limit is set to 20000 clock cycles for the unidirectional Fano decoder as Ran and others (2009). But decoding delay of the decoder designed is expected to be greater than the simulation results mentioned by Ran and others. The difference is caused by the *look back* step in the Fano algorithm. If M_B is greater or equal to the threshold in more than one clock cycle, the decoder moves backward consecutively. The algorithm mentioned by Ran and others counts these consecutive moves as a single iteration, but in hardware the decoder can move only one time in a clock cycle.

The overflow limit of the bidirectional Fano decoder is set to 10000 clock cycles. Because the forward and the backward decoders make two iterations in a single clock cycle. So the overflow limit of the bidirectional Fano decoder should be half of the overflow limit of the unidirectional Fano decoder.

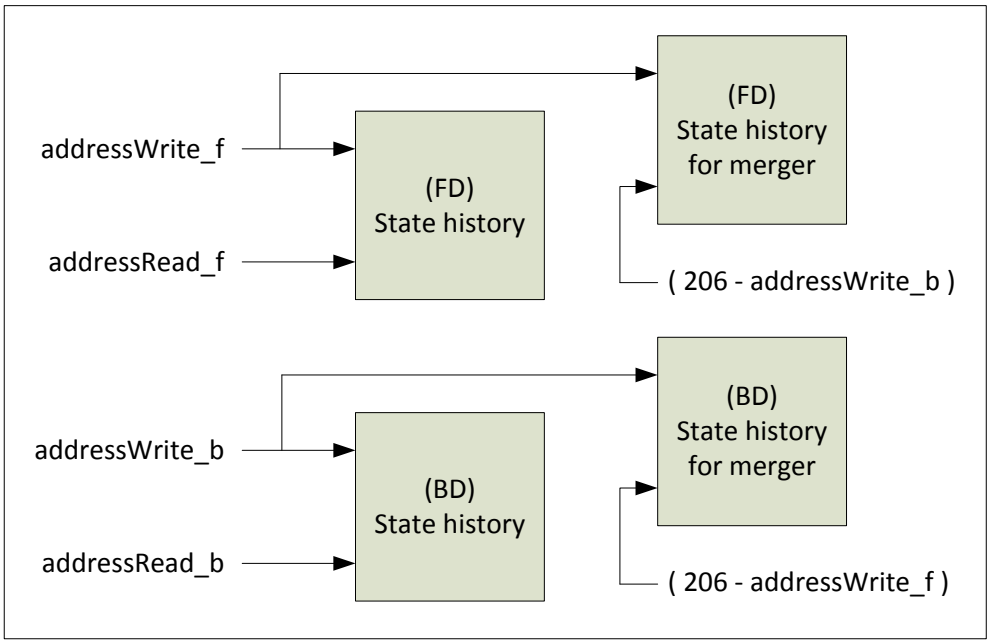
4.1.6 Modifications for Bidirectional Decoding

The bidirectional Fano decoder is very similar to the unidirectional Fano decoder. Basically two unidirectional decoders work in parallel. Additionally merging logic exists and there are extra buffers for state history. There are also minor differences between FD and BD.

The FD uses reverse of the generator polynomials (see chapter 2.2) as in the unidirectional Fano decoder, but the BD uses the same polynomials as the encoder.

Another difference is RAM addressing. The FD starts reading the input buffer from address zero, the BD starts from 205 which is the address of the latest symbol. The FD writes to the output buffer by starting from address zero again, but The BD starts from 199. History buffers also are driven by starting from address 205 by the BD.

Figure 4.6: Buffers for state history



Each one of the FD and the BD uses two buffers for state history. One buffer is for regular usage and another is for checking merger. Same data is written to both buffers. The second buffer of the FD is controlled by the BD and the second buffer of the BD is controlled by the FD. The states read from these buffers are compared. Equivalence of only one state is enough for merger in the Fano decoder designed. Figure 4.6 shows how the buffers are addressed and controlled.

4.1.7 Merging Depth

The bidirectional Fano decoder designed does not combine outputs of the FD and the BD into a separate buffer. It keeps the merging depth in a register after a codeword is decoded. Then it sends the output outside by reading the codeword from output buffer of the FD and that of the BD. Same control signals are sent to these two buffers, but outputs of the buffers are multiplexed. While address of the buffers is smaller then the merging depth output of the FD is selected, else output of the BD is selected.

This structure is designed for the simulation environment mentioned in chapter 4.2. The decoder does not decode any codeword while sending the output. To avoid this problem, the output buffers can be doubled. The decoder writes to one pair of the buffers while other buffers are sending the output.

4.1.8 BER Improvement for Bidirectional Decoding

Since average number of iterations (*NoI*) per symbol is expected to be low at SNR=6, a simple modification can be made to improve BER. The bidirectional Fano decoder designed checks only one state for merging, but the states can be checked in two consecutive clock cycles, then FD and BD merge if the states are equivalent in both clock cycles.

At low SNR values, merging of the decoders may be too difficult by reason of backward moves and decoding delay may increase too much. So this modification is made only at SNR=6.

4.2 SIMULATION ENVIRONMENT

Hardware implementation of the Fano decoder is based on an algorithm written in Matlab language. However the results of the hardware and that of the algorithm are not compared one to one, some parts of the environment are composed by the Matlab algorithm. For example, the additive white Gaussian noise (*AWGN*) channel is not implemented in hardware. Because it is not a part of the design, it is required for simulations only.

Figure 4.7: Simulation setup

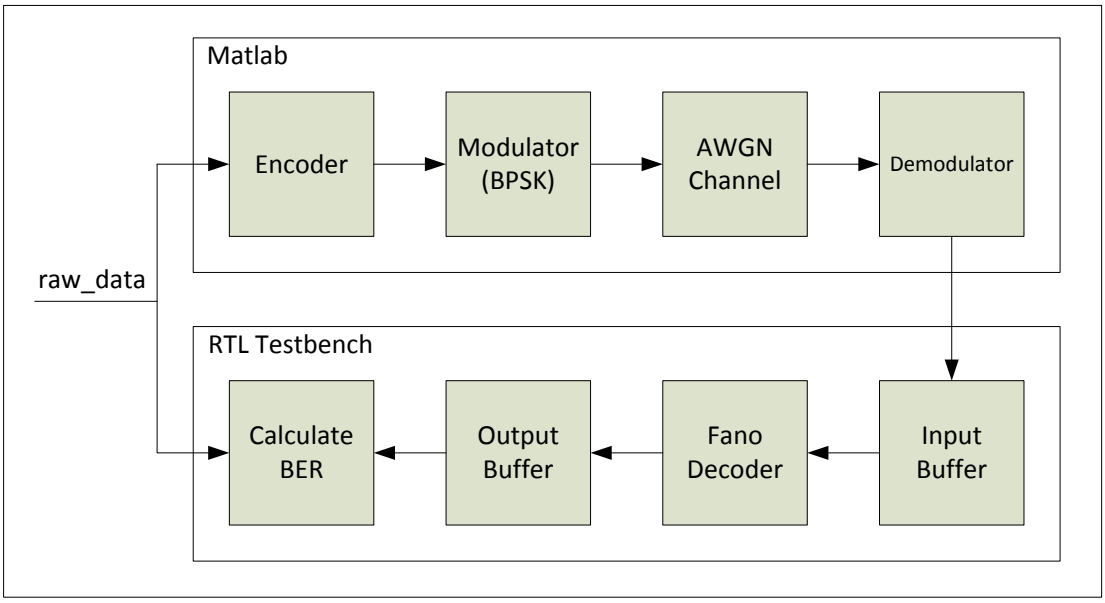


Figure 4.7 shows the simulation setup. Matlab algorithm generates a random codeword whose *L* equals to 200. The codeword is encoded by a convolutional encoder which is introduced in Figure 2.1.

The encoded data is modulated by the digital modulation technique, binary phase-shift keying (*BPSK*). Then the modulated data passes through the AWGN channel. Some noise is applied in this channel by considering the SNR entered. The SNR values used in the simulations are 3, 4, 5 and 6 in dB.

Then the noisy data is demodulated. The raw data and the demodulated data are fed to the RTL testbench which is mentioned in chapter 4.2.1 in detail. The Matlab algorithm also calculates and provides some metrics which depend on the SNR value.

4.2.1 RTL Testbench

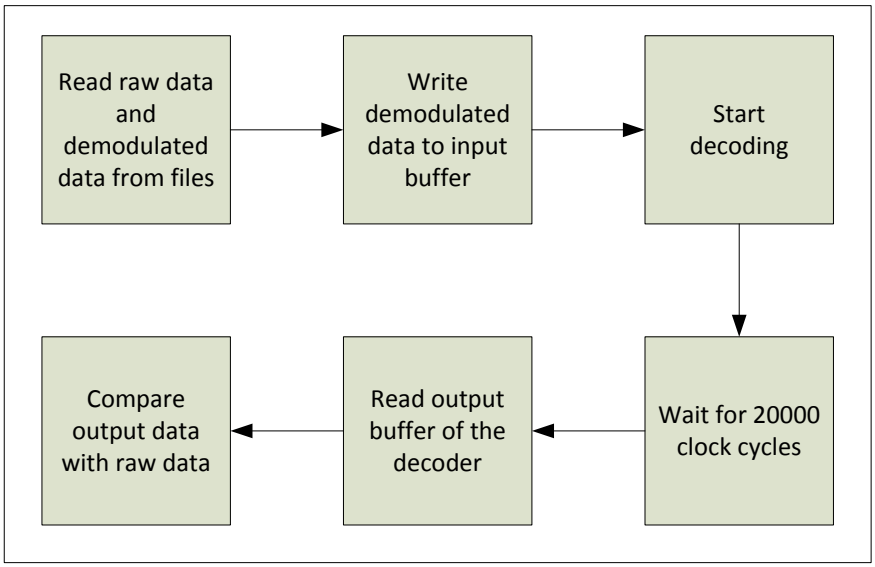
The RTL testbench written in Verilog reads two text files that include the raw data and the encoded (noisy and demodulated) data provided by the Matlab algorithm. For a raw data file, there are four encoded data files. These files are for different SNR values. Unidirectional and bidirectional Fano decoders are simulated concurrently but each simulation runs for only one SNR and delta value.

Encoded data is written to the input buffers. After writing, the testbench sends a message as a pulse to the decoders to make them start decoding. The decoders send a message back when they finished decoding. Then the testbench reads the output buffers and compares the data in buffers with the original (raw) data read from the text file. The comparison is not performed right after the decoding finishes. The testbench always waits until the overflow time expires.

After comparison, the testbench logs results of the simulation to a text file. The results obtained from a simulation are as follows:

- Decoding delay in terms of clock cycle
- Number of erroneous bits in decoded data
- Maximum values of some metrics

Figure 4.8: Flowchart of testbench

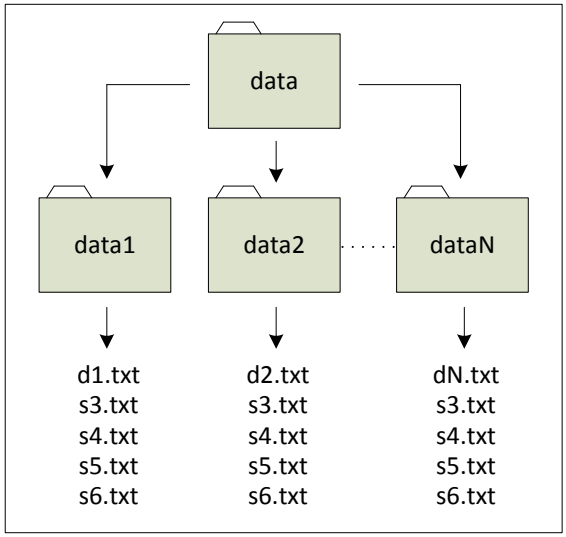


The flowchart in Figure 4.8 shows how the testbench works. This flow is repeated for each SNR value, so one raw data set is simulated four times for a specific delta value. All simulations are controlled by a TCL script. After a simulation finishes, next simulation starts automatically until all data sets are simulated for all SNR values.

4.2.2 Preparing Data Sets

Data sets were prepared by using the Matlab algorithm. The algorithm runs in a loop and generates 500 random codewords which include raw data. Also four encoded codewords are generated for each raw data for different SNR values.

Figure 4.9: Directory structure of data



Output of the Matlab algorithm is processed by a Perl script and the data is written in text files which are read by the RTL testbench. Directory structure of the files is shown in Figure 4.9. There are five files in a folder named as *data(i).* File *d(i).txt* contains raw data and file *s(j).txt* contains encoded data where j is the SNR value.

4.3 SYNTHESIS ENVIRONMENT

The decoders designed were synthesized by using Xilinx ISE. One target FPGA is XC6SLX9. It is one of the smallest FPGAs of Spartan-6 family. Synthesis results for this FPGA is used for comparison of the Fano decoder and the Viterbi decoder. Another target FPGA is XC7K70T which belongs to Kintex-7 family. It is the smallest FPGA of Xilinx 7-series FPGAs. Synthesis for XC7K70T aims to show what clock frequency can be achieved in one of the latest FPGA families. Properties of the two FPGAs used in thesis will be elaborated in chapter 4.3.6.

4.3.1 Configuration of Fano Decoder

Width of some signals was determined after behavioral simulations, because these signals do not have a theoretical limit. For example, the path metric and the threshold value do not exceed 512 according to the simulations. Since they are signed numbers, their width must be at least 10-bit.

The Fano decoder has some hard coded parameters for different SNR values. Synthesis results may vary due to these parameters, but the difference is too small. However SNR value is set to 6 and the delta is set to 2 during synthesis, it is assumed that the synthesis results are equivalent for all configurations.

4.3.2 User Constraints

Input/output signals of the decoder are not locked to any FPGA pins. Xilinx ISE determines the pins itself. Default temperature and voltage values provided by ISE are used. The temperature is 85°C and the voltage is 1.14V. Clock frequency is the only synthesis constraint that is set in a user constraint file (*UCF*). For example, a 43 MHz clock is set as follows.

NET "clk" TNM_NET = clk;
TIMESPEC TS_clk = PERIOD "clk" 23.255 ns HIGH 50%;

Critical paths of the unidirectional and bidirectional Fano decoders are common. If one of the decoders reaches the target clock frequency, it is assumed that the other decoder also reaches the target.

4.3.3 Synthesized Architectures

Two architectures of the Fano decoder are synthesized. The architecture that drives the buffers with *clk_2x* is called the first architecture and the other architecture that assigns signals of the buffers without flip-flops is called the second architecture as mentioned in chapter 4.1.3.

The first architecture is used in simulations also. So this architecture is verified sufficiently. However the second architecture has different synthesis results with respect to the first architecture, it is expected to give the same simulation results. After a few simulations it gave the same output (decoded) codewords, but the decoding delay was a little bit different. So the designs in this architecture may have some minor bugs that are not expected to affect the synthesis results too much.

However maximum clock frequency achieved in the second architecture will be used while throughput is being calculated, FPGA resources consumed will be used from the first architecture. Theoretically the second architecture consumes less flip-flops, also

synthesis results verified this. But results of the second architecture is not too reliable, so results of the first architecture can be assumed as worst case. It can be thought that the most important aim of the second architecture is seeing maximum clock frequency.

Since critical paths of the unidirectional and bidirectional Fano decoders are common, the second architecture is applied to the unidirectional Fano decoder only. As mentioned in chapter 4.3.2, maximum clock frequency achieved will be used for both decoders.

4.3.4 Calculation of Throughput

Throughputs of the Fano decoders are calculated by equation (4.1) after the simulations and synthesis. The equation is as follows:

$$T = \frac{L}{D} \cdot f \tag{4.1}$$

where T is throughput, L is codeword length, D is decoding delay in terms of clock cycle and f is clock frequency. If f is in terms of MHz, T will be in terms of Mbps.

4.3.5 Calculation of Throughput/Area

Number of LUT/FF pairs is used as if it represents area of a decoder. Thoughput of the Viterbi decoder is divided by 2601 which is the number of LUT/FF pairs and the result is multiplied by a constant to equalize Viterbi's ratio to one. Since the throughput is 126 MHz, the constant equals to 20.64. Then throughput/area ratios of all decoders are multiplied by this constant. This calculation is shown in equation (4.2).

$$Ratio = \frac{Throughput}{N} \times 20.64 \tag{4.2}$$

where N is the number of LUT/FF pairs.

4.3.6 Properties of FPGAs

XC6SLX9 has 1430 slices. Since each slice contains four LUTs and eight flip-flops in all Spartan-6 FPGAs, there are 5720 LUTs and 11440 flip-flops in XC6SLX9. Slices can be configured as RAM too. If all slices are used as RAM, XC6SLX9 can provide 90 Kb memory in maximum. RAMs implemented in slices are called distributed RAM by Xilinx Inc. In spite of the distributed RAMs, in the FPGA there are also hard RAM blocks which are called block RAM. In XC6SLX9, there are 32 block RAMs and each

RAM (RAMB16BWER) is 18 Kb. One RAM can be used as two separate 9 Kb RAMs (RAMB8BWER) too. Total size of block RAMs is 576 Kb. For clock management, two PLLs and four DCMs exist in XC6SLX9. Same FPGA can have different device packages, XC6SLX9 has five packages. The package which has minimum user I/Os is chosen for synthesis. The package is TQG144 and it has 102 user I/Os. Speed grade is chosen -3 which is the fastest grade for XC6SLX9.

XC7K70T has 10250 slices. Each 7 series FPGA slice contains four LUTs and eight flip-flops, so there are 41000 LUTs and 82000 flip-flops in XC7K70T. If slices are configured as distributed RAM, 838 Kb memory is provided in maximum. There are 135 block RAMs (RAMB36E1) and each one provides 36 Kb memory. Each 36 Kb block RAM can be used as two separate 18 Kb RAMs (RAMB18E1). Total size of block RAMs is 4860 Kb. In XC7K70T, there are six PLLs and six mixed mode clock managers (*MMCM*). In 7 series FPGAs, MMCM exists instead of DCM. XC7K70T has two device packages. FBG484 is chosen and it has 285 user I/Os. Speed grade of XC7K70T is chosen -3 as XC6SLX9.

5. EXPERIMENTAL RESULTS

5.1 SIMULATION RESULTS

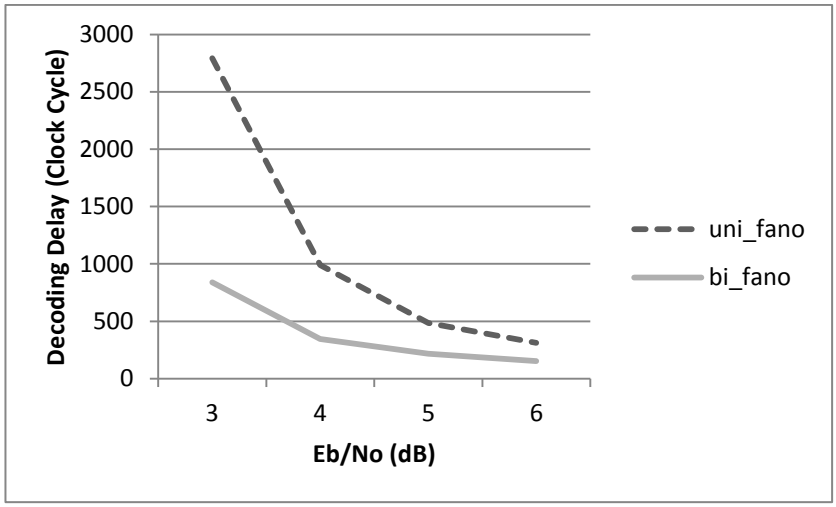
500 codewords were simulated in ModelSim. The codewords which had reached the overflow limit were excluded from the results and they were not counted while the average decoding delay and BER were being calculated. A codeword was excluded from both unidirectional and bidirectional decoding results, if it reaches the limit during any of the decoding techniques. But the codeword was not excluded for all SNR and delta values.

At first, delta was set to eight. No codeword reached the overflow limit while SNR was five or six. Number of codewords which reached the limit was one while SNR equaled to four and the number was 13 while SNR equaled to three. Most of the codewords that reached the limit were observed during unidirectional decoding and only one codeword reached the overflow limit during bidirectional decoding while SNR was three.

5.1.1 Decoding Delay

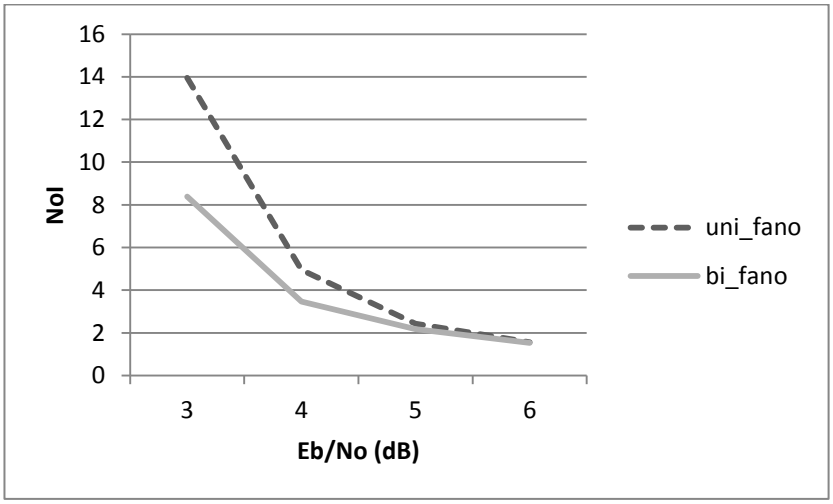
Figure 5.1 shows the average decoding delay of the unidirectional and bidirectional decoders while delta equals to eight. Average decoding delay results of the unidirectional Fano decoder at SNR from three to six are 2792, 989, 485 and 312 clock cycles respectively. Results of the bidirectional Fano decoder are 839, 346, 217 and 153 clock cycles.

Figure 5.1: Average decoding delay at delta=8



If the delay values given above are divided by 200 which is the codeword length, *NoI* per symbol is found. We should note that the bidirectional Fano decoder makes two parallel iterations in a clock cycle. The result may be multiplied by two. But in some calculations, decoding delay in terms of clock cycles will be used rather than *NoI*. Parallel iterations affect the area of the decoders, so this concern will be considered when throughput/area is calculated. *NoI* per symbol is shown in Figure 5.2 where parallel iterations are considered.

Figure 5.2: NoI per symbol at delta=8

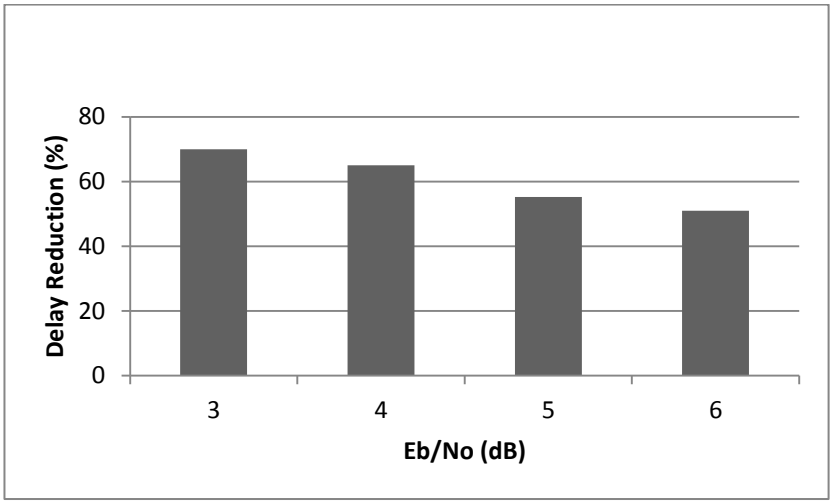


NoI per symbol of the unidirectional Fano decoder (*uni_fano*) is 13.96 and *NoI* of the bidirectional Fano decoder (*bi_fano*) is 8.39 at SNR=3dB. Then *NoI* of *uni_fano* equals

to 4.95, 2.43 and 1.56 at higher SNR values respectively. *NoI* of *bi_fano* is 3.46 at SNR=4dB, 2.17 at SNR=5dB and 1.53 at SNR=6 dB.

Figure 5.3 shows how much the bidirectional Fano decoder reduces the decoding delay in terms of clock cycles when compared with the unidirectional Fano decoder. The reduction is 69.9% at SNR=3dB, 65% at SNR=4dB, 55.3% at SNR=5dB and 51% at SNR=6dB. Again these results are valid at delta=8.

Figure 5.3: Decoding delay reduction by using bi_fano at delta=8



However maximum and minimum decoding delays do not give much information, they give minor clues about computational variability. At SNR=6dB, maximum decoding delay of the unidirectional Fano decoder is 3326 clock cycles and that of the bidirectional one is 337 clock cycles. Table 5.1 gives whole results.

Table 5.1: Maximum and minimum decoding delay at delta=8

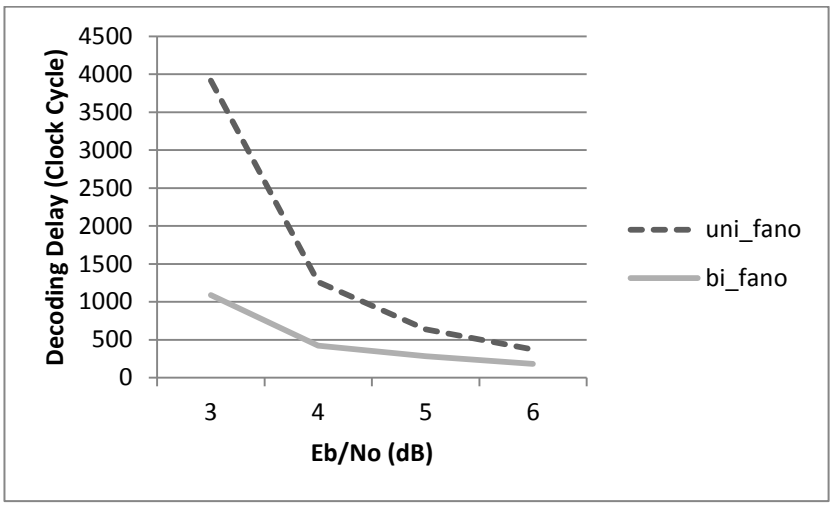
	3		4		5		6	
	Uni	Bi	Uni	Bi	Uni	Bi	Uni	Bi
Maximum*	18292	5287	7826	1388	5705	672	3326	337
Minimum	359	182	229	126	232	117	218	110

* Overflow limits are ignored.

After the modification mentioned in chapter 5.1.8 is applied, average decoding delay of the bidirectional Fano decoder increases from 153 to 159 clock cycles. Maximum delay increases from 337 to 481 and minimum delay increases from 110 to 111.

The unidirectional and bidirectional Fano decoders were also simulated at delta=4. Again no codeword reached the overflow limit at SNR=5 and at SNR=6. The unidirectional Fano decoder reached the limit one time at SNR=4 and 22 times at SNR=3. The bidirectional Fano decoder reached the limit 3 times only at SNR=3. Average decoding delays at delta=4 are shown in Figure 5.4.

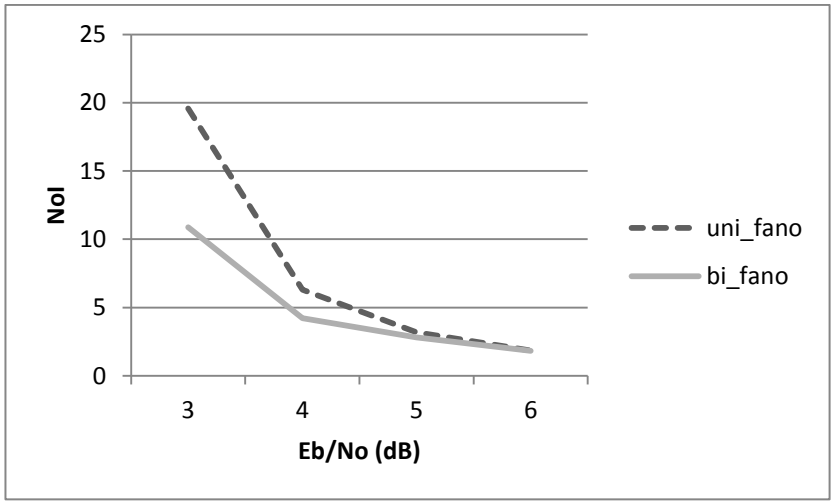
Figure 5.4: Average decoding delay at delta=4



Decoding delay of the unidirectional Fano decoder is 3916 clock cycles at SNR=3dB and decoding delay of the bidirectional Fano decoder is 1087 clock cycles. At higher SNR values, delay of *uni_fano* is 1262, 636 and 372 respectively. Delay of *bi_fano* is 422, 282 and 182 clock cycles.

Figure 5.5 shows *NoI* per symbol. Again parallel iterations are considered as in Figure 5.2.

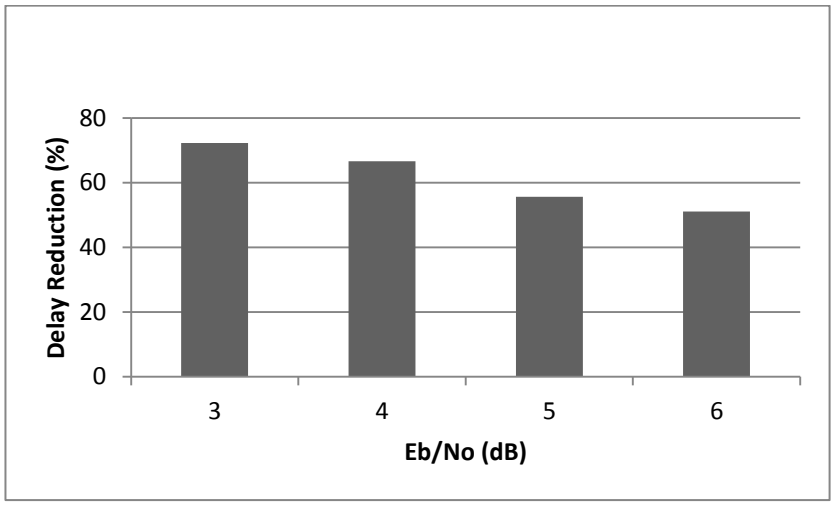
Figure 5.5: NoI per symbol at delta=4



At SNR=3dB, *NoI* per symbol of *uni_fano* is 19.58 and that of *bi_fano* equals to 10.87. *NoI* of *uni_fano* is 6.31, 3.18 and 1.86 at SNR values from 4 to 6 respectively. *NoI* of *bi_fano* is 4.22, 2.82 and 1.82 at the same SNR values.

Figure 5.6 shows how much the bidirectional Fano decoder reduces the average decoding delay with respect to the unidirectional Fano decoder at delta=4.

Figure 5.6: Decoding delay reduction by using bi_fano at delta=4



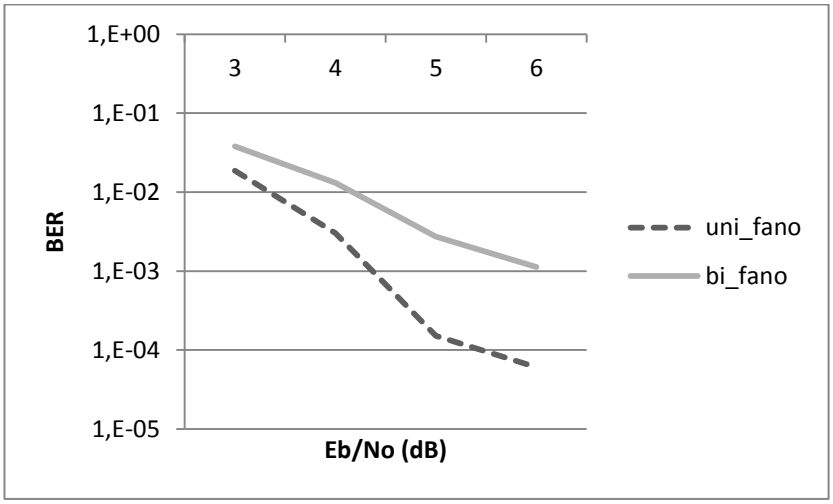
The bidirectional Fano decoder reduces the decoding delay 72.2% with respect to the unidirectional Fano decoder at SNR=3dB. It reduces 66.6% at SNR=4dB, 55.7% at SNR=5dB and 51.1% at SNR=6dB.

5.1.2 Bit Error Rate

Each simulation gives bit error information in number of erroneous bits in a codeword whose length is 200. Results of all simulations are summed and divided by 500 which is the number of simulations. This operation gives average number of erroneous bits in a codeword. Then this value is divided by 200 and percentage of error is found.

The logarithmic SNR vs. BER graph is shown in Figure 5.7. These results are valid at delta=8 as the average decoding delays in Figure 5.1.

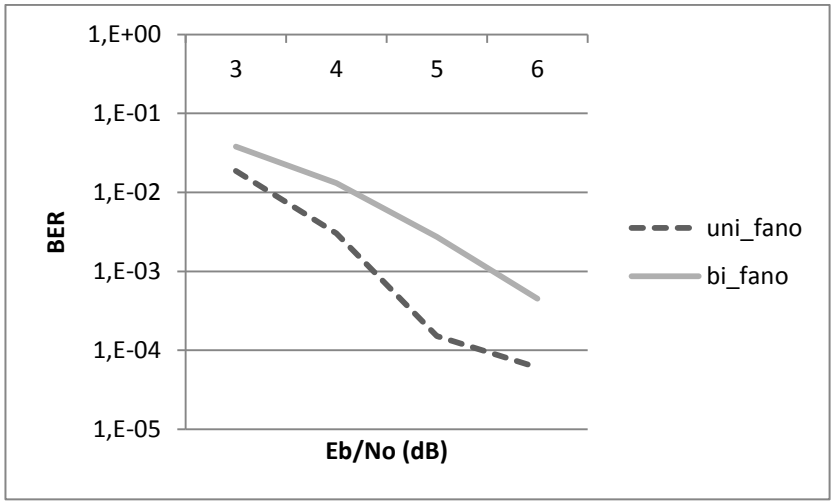
Figure 5.7: SNR vs. BER graph at delta=8



BER of the *uni_fano* is 18.63×10^{-3} and BER of the *bi_fano* is 37.9×10^{-3} at SNR=3dB. BER of *uni_fano* is 3.04×10^{-3} at SNR=4dB, 1.5×10^{-4} at SNR=5dB and 0.6×10^{-4} at SNR=6dB. BER of *bi_fano* is 13.05×10^{-3}, 2.73×10^{-3} and 1.12×10^{-3} respectively.

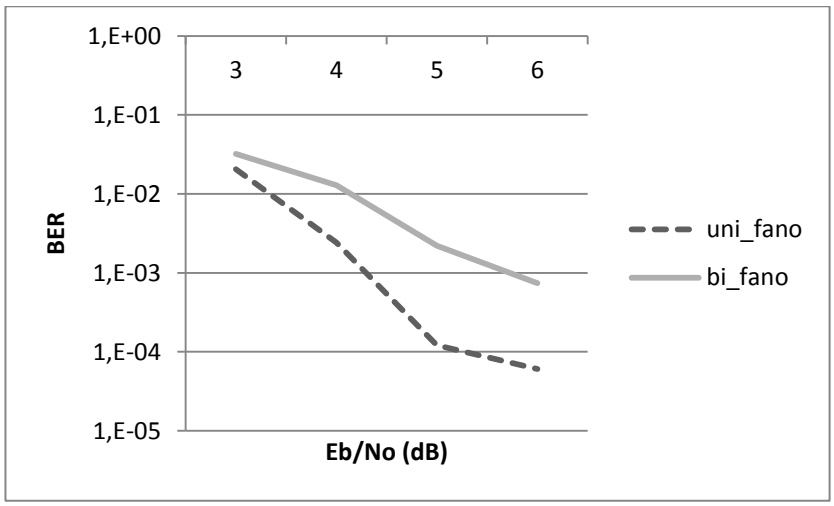
The modification mentioned in chapter 5.1.8 reduces BER from 1.12×10^{-3} to 4.5×10^{-4} at SNR=6dB. After this improvement Figure 5.7 can be updated as shown in Figure 5.8.

Figure 5.8: SNR vs. BER graph at delta=8 after improvement



SNR vs. BER graph in Figure 5.9 shows the results at delta=4. These results do not include the improvement at SNR=6dB.

Figure 5.9: SNR vs. BER graph at delta=4



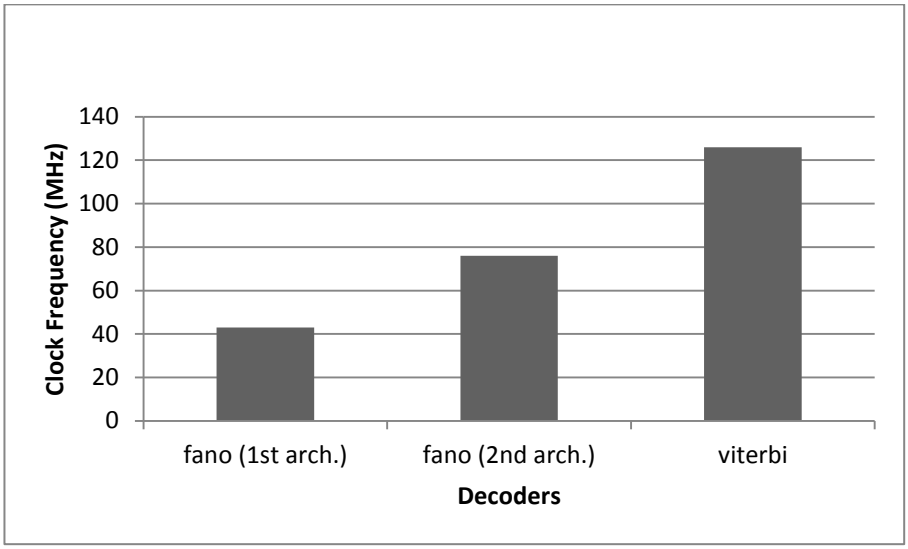
BER of the unidirectional Fano decoder is 20.47×10^{-3} at SNR=3dB, 2.43×10^{-3} at SNR=4dB, 1.2×10^{-4} at SNR=5dB and 0.6×10^{-4} at SNR=6dB. BER of the bidirectional Fano decoder is 32.24×10^{-3}, 12.9×10^{-3}, 2.2×10^{-3} and 7.4×10^{-4} respectively.

5.2 SYNTHESIS RESULTS

5.2.1 Clock Frequency

There are two different architectures of the Fano decoder as mentioned in chapter 4.1.3 and 4.3.3. The first architecture of the unidirectional and bidirectional Fano decoders work at 43 MHz clock frequency in Spartan-6 FPGAs. The second architecture works at 76 MHz. The parallel version of the Viterbi decoder IP works at 126 MHz in maximum.

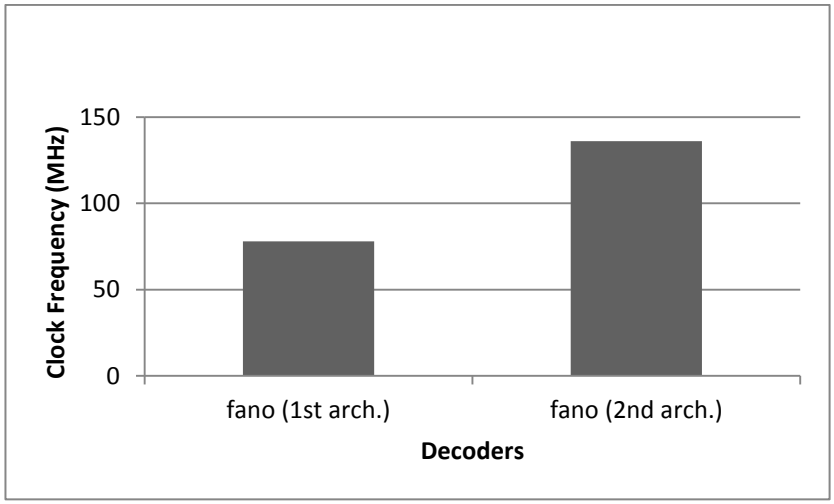
Figure 5.10: Clock frequency in Spartan-6



That should be noted that the Viterbi decoder IP does not take jitter into account, but the first architecture of the Fano decoder does. In this architecture, the clocks (*clk* and *clk_2x*) are derived from an external clock in a PLL and ISE specifies a default clock jitter. In Spartan-6 FPGAs, total clock uncertainty of the first architecture is 0.236 ns. Since there is no PLL in the second architecture, input jitter is ignored. Total clock uncertainty equals to 0.035 ns in Spartan-6.

In Kintex-7 FPGAs, the first architecture of the Fano decoder works at 78 MHz and the second architecture works at 136 MHz. The clock uncertainty of the first architecture is 0.181 ns and that of the second architecture equals to 0.035 ns as in Spartan-6. The Viterbi decoder IP is not available for Kintex-7.

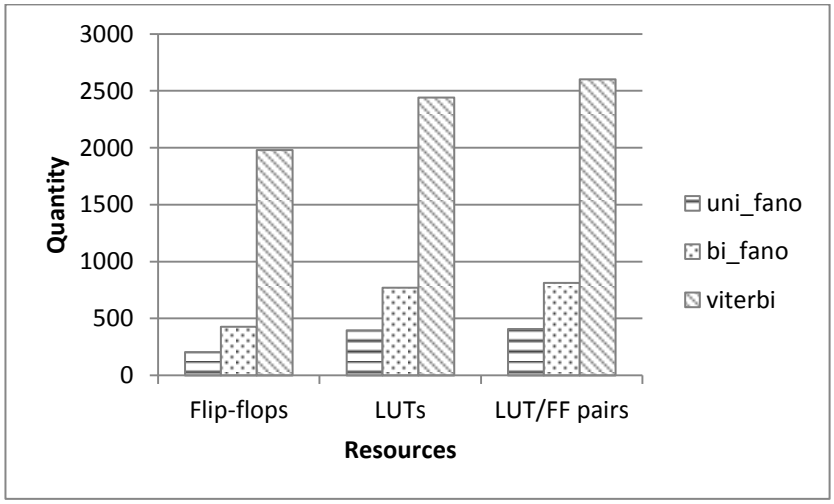
Figure 5.11: Clock frequency in Kintex-7



5.2.2 FPGA Resources

FPGA resources consumed by the unidirectional and bidirectional Fano decoders and the Viterbi decoder IP in Spartan-6 FPGAs are shown in Figure 5.12. These results of the Fano decoders are available for only the first architecture.

Figure 5.12: FPGA resources consumed in Spartan-6



The unidirectional Fano decoder consumes 205 flip-flops, the bidirectional Fano decoder does 428 and the Viterbi decoder does 1980 flip-flops. Number of flip-flops that *uni_fano* consumes is 47.9% of bi_fano and 10.4% of the Viterbi decoder. *Bi_fano* consumes 21.6% of the Viterbi decoder. *Uni_fano* consumes 398 LUTs, *bi_fano* 770

and the Viterbi decoder 2442. That means *uni_fano* consumes 51.7% of *bi_fano* and 16.3% of the Viterbi decoder. *Bi_fano* consumes 31.5% of the Viterbi decoder.

After synthesis, LUT/FF pairs show the number of pairs which at least a flip-flop or a LUT is consumed. So the number is always greater than the number of flip-flops and that of LUTs. *Uni_fano* consumes 407 pairs, *bi_fano* 812 and the Viterbi decoder 2601. LUT/FF pairs that *uni_fano* consumes are 50.1% of *bi_fano* and 15.6% of the Viterbi decoder. *Bi_fano* consumes 31.2% of the Viterbi decoder.

In addition, *uni_fano* consumes five 9 kb RAMs (RAMB8BWER) and *bi_fano* consumes 11 RAMs. The Viterbi decoder consumes two 18 kb RAMs (RAMB16BWER) that can be thought as four 9 kb RAMs.

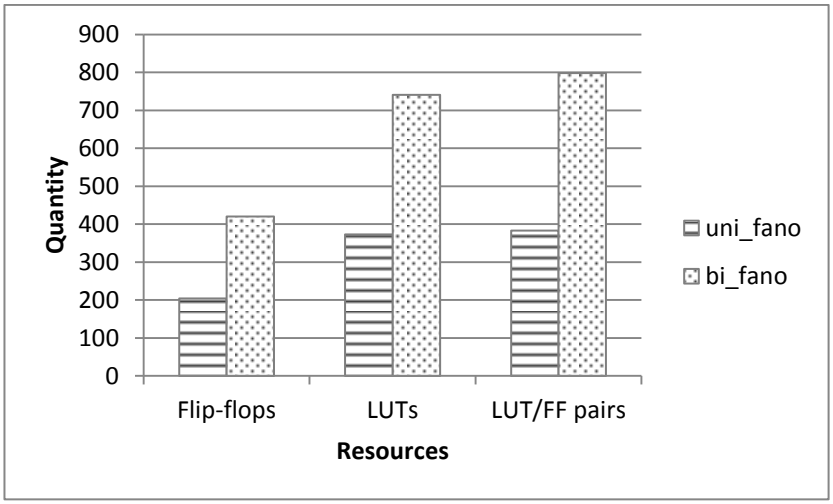
What percent of resources consumed in XC6SLX9 is as follows. *Uni_fano* consumes 1.8% of flip-flops, 7% of LUTs and 7.8% of RAMs. *Bi_fano* consumes 3.7% of flip-flops, 13.5% of LUTs and 17.2% of RAMs. The Viterbi decoder consumes 17.3% of flip-flops, 42.7% of LUTs and 6.3% of RAMs.

The first architecture of the Fano decoder also uses a PLL to generate *clk_2x*. This is valid for both unidirectional and bidirectional Fano decoders.

However FPGA resources consumed in the second architecture are not used in calculations as mentioned in chapter 4.3.3, we can note that the unidirectional Fano decoder consumes 97 flip-flops in this architecture. In other words, it consumes 47.3% of the first architecture. It can be assumed that the bidirectional Fano decoder also consumes about 47% of the flip-flops with respect to the first architecture.

Results of the first architecture are also available for Kintex-7. These results are shown on the graph in Figure 5.13.

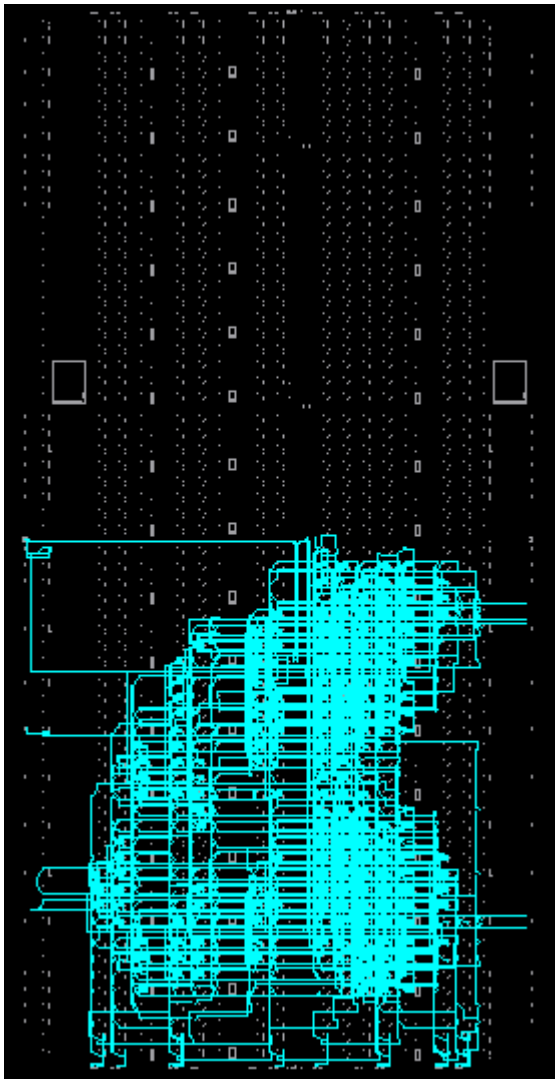
Figure 5.13: FPGA Resources consumed in Kintex-7



Uni_fano consumes 204 flip-flops and 373 LUTs. *Bi_fano* consumes 420 flip-flops and 741 LUTs. Number of flip-flops consumed by *uni_fano* is 48.6% of bi_fano and number of LUTs is 50.3% of *bi_fano*. *Bi_fano* consumes 798 LUT/FF pairs, *uni_fano* consumes 383 pairs that means 48% of *bi_fano*. *Uni_fano* consumes five 18 Kb RAMs (RAMB18E1) and *bi_fano* consumes 11 RAMs. What percent of resources consumed in XC7K70T is as follows. *Uni_fano* consumes 0.25% of flip-flops, 0.91% of LUTs and 1.9% of RAMs. *Bi_fano* consumes 0.51% of flip-flops, 1.8% of LUTs and 4.1% of 18 Kb RAMs (RAMB18E1).

FPGA resources consumed by the bidirectional Fano decoder are shown in Figure 5.14. Actually, the cyan lines show routing of the circuit. This screenshot is captured from FPGA Editor which is a tool in ISE Design Suite and it shows whole of the FPGA. The circuit belongs to the first architecture of the bidirectional Fano decoder. It is synthesized in the Spartan-6 FPGA (XC6SLX9).

Figure 5.14: Screenshot from FPGA Editor

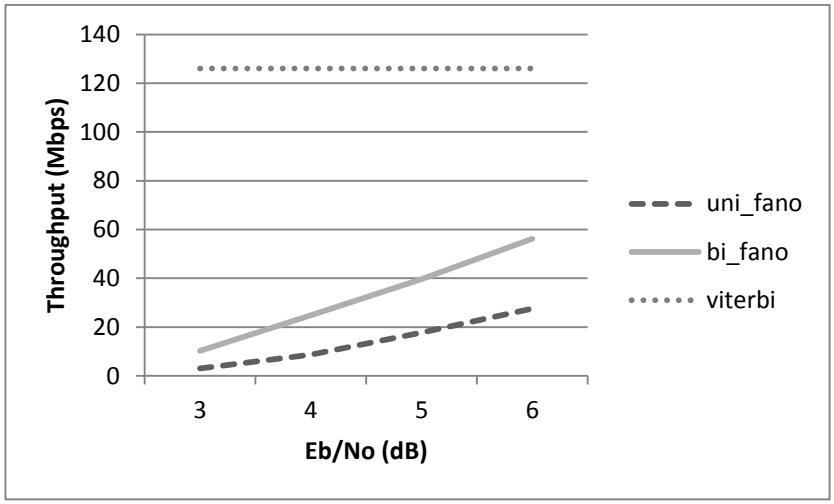


5.2.3 Throughput

Throughput of the Fano decoder is calculated as shown in equation (4.1). Decoding delays at delta=8 are used for the two architectures and also for the two FPGAs. Only the clock frequencies are different. Throughput of the Viterbi decoder IP is 126 Mbps as mentioned in its datasheet (Viterbi Decoder v7.0). The parallel version of the Viterbi decoder gives one output in each clock cycle, so its throughput only depends on the clock frequency which is 126 MHz is in Spartan-6 FPGAs.

Throughput results of the Fano decoders for the first architecture and the Viterbi decoder in Spartan-6 FPGAs are shown in Figure 5.15. Clock frequency of the Fano decoders is 43 MHz as mentioned in chapter 5.2.1.

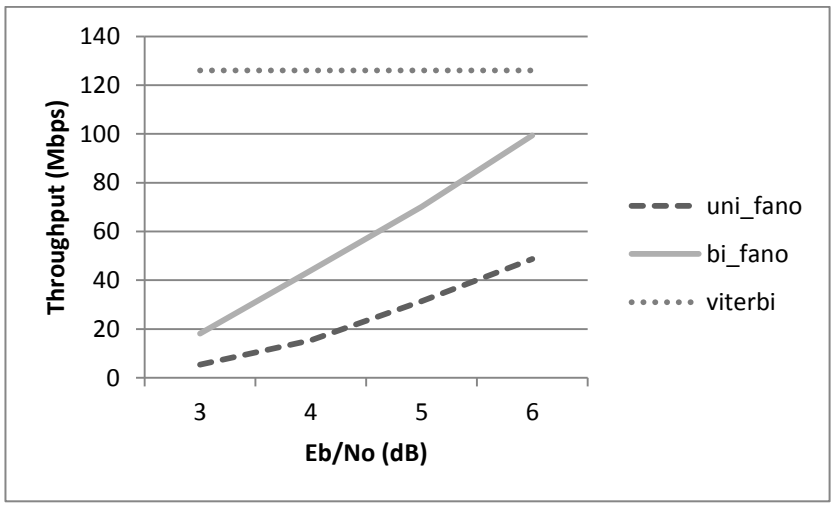
Figure 5.15: Throughput for 1st architecture in Spartan-6



While the Viterbi decoder works at 126 Mbps at all SNR values, *uni_fano* works at 3.1 Mbps and *bi_fano* works at 10.3 Mbps at SNR=3dB. At other SNR values, *uni_fano* works at 8.7, 17.7 and 27.6 Mbps respectively. *Bi_fano* works at 24.9, 39.6 and 56.2 Mbps.

Throughput results of the Fano decoders for the second architecture and the Viterbi decoder in Spartan-6 FPGAs are shown in Figure 5.16. Clock frequency is 76 MHz.

Figure 5.16: Throughput for 2nd architecture in Spartan-6

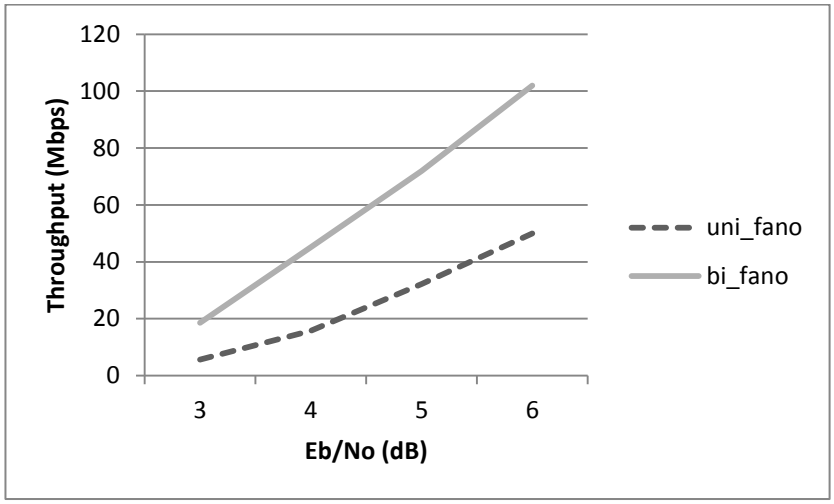


Throughput of the unidirectional Fano decoder is 5.4 Mbps at SNR=3dB, 15.4 Mbps at SNR=4dB, 31.3 Mbps at SNR=5dB and 48.7 Mbps at SNR=6dB. Throughput of the

bidirectional Fano decoder is 18.1 Mbps at SNR=3dB, 43.9 Mbps at SNR=4dB, 70 Mbps at SNR=5dB and 99.3 Mbps at SNR=6dB.

Throughput results of the Viterbi decoder are not available for Xilinx 7-series FPGAs in its datasheet. Throughput results of the unidirectional and bidirectional Fano decoders for the first architecture in Kintex-7 FPGAs are shown in Figure 5.17. Clock frequency equals to 78 MHz.

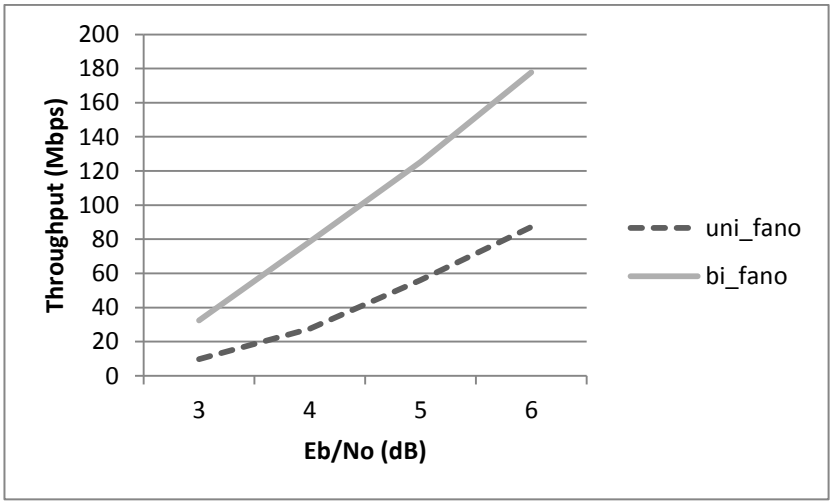
Figure 5.17: Throughput for 1st architecture in Kintex-7



Throughput of the unidirectional Fano decoder is 5.6 Mbps at SNR=3dB, 15.8 Mbps at SNR=4dB, 32.2 Mbps at SNR=5dB and 50 Mbps at SNR=6dB. Throughput of the bidirectional Fano decoder is 18.6 Mbps at SNR=3dB, 45.1 Mbps at SNR=4dB, 71.9 Mbps at SNR=5dB and 102 Mbps at SNR=6dB.

Throughput results of the Fano decoders for the second architecture in Kintex-7 FPGAs are shown in Figure 5.18. Clock frequency equals to 136 MHz.

Figure 5.18: Throughput for 2nd architecture in Kintex-7

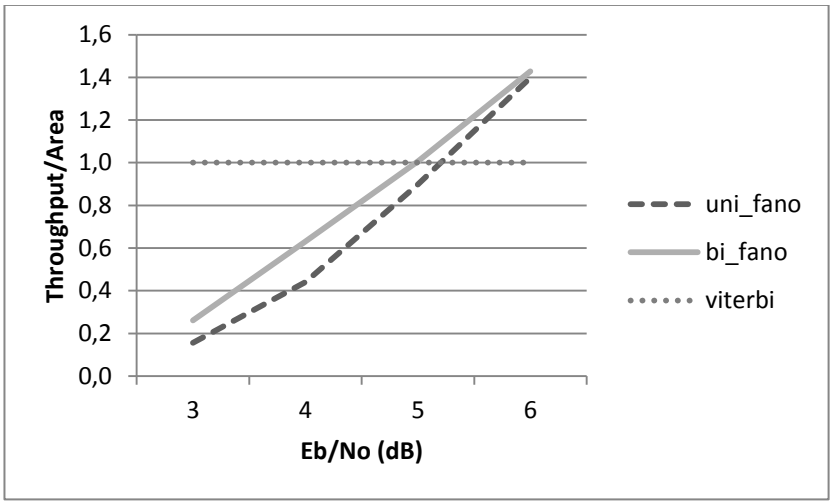


Throughput of the unidirectional Fano decoder is 9.7 Mbps at SNR=3dB, 27.5 Mbps at SNR=4dB, 56.1 Mbps at SNR=5dB and 87.2 Mbps at SNR=6dB. Throughput of the bidirectional Fano decoder is 32.4 Mbps at SNR=3dB, 78.6 Mbps at SNR=4dB, 125.3 Mbps at SNR=5dB and 177.8 Mbps at SNR=6dB.

5.2.4 Throughput/Area

Throughput/area is calculated by equation (4.2). Throughputs in Spartan-6 FPGAs are used to be able to compare the Viterbi decoder with the unidirectional and bidirectional Fano decoders. Results for the first architecture are shown in Figure 5.19.

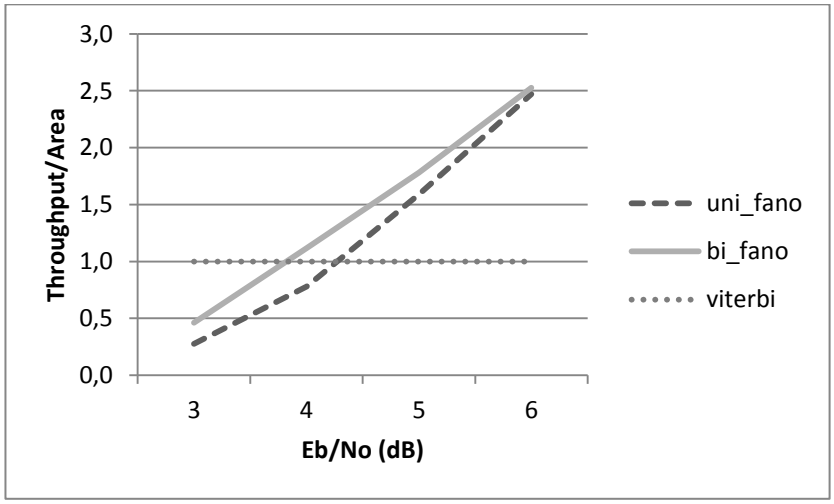
Figure 5.19: Throughput/area for 1st architecture



Ratio of the Viterbi decoder equals to one as mentioned in chapter 4.3.5. Ratio of the unidirectional Fano decoder equals to 0.16 at SNR=3dB, 0.44 at SNR=4dB, 0.9 at SNR=5dB and 1.4 at SNR=6dB. Ratio of the bidirectional Fano decoder equals to 0.26 at SNR=3dB, 0.63 at SNR=4dB, 1.01 at SNR=5dB and 1.43 at SNR=6dB.

Results for the second architecture are shown in Figure 5.20.

Figure 5.20: Throughput/area for 2nd architecture



Ratio of the unidirectional Fano decoder equals to 0.28 at SNR=3dB, 0.78 at SNR=4dB, 1.59 at SNR=5dB and 2.47 at SNR=6dB. Ratio of the bidirectional Fano decoder equals to 0.46 at SNR=3dB, 1.12 at SNR=4dB, 1.78 at SNR=5dB and 2.53 at SNR=6dB.

Figure 5.21: Throughput/area improvement of bi_fano

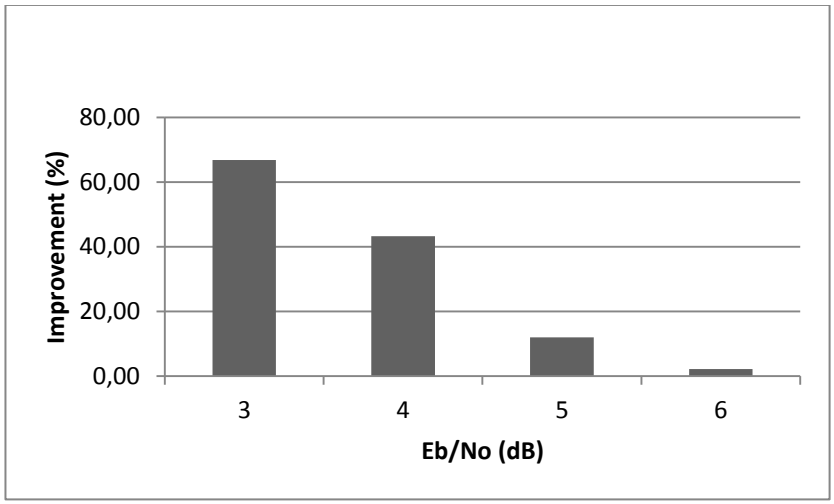


Figure 5.21 shows how much the bidirectional Fano decoder improves throughput/area ratio when compared with unidirectional Fano decoder. The improvement is 66.8% at SNR=3dB, 43.27% at SNR=4dB, 12.03% at SNR=5dB and 2.21% at SNR=6dB.

6. CONCLUSION

The unidirectional and bidirectional Fano decoders were designed in two architectures. Pipelining the Fano decoder is not easy. Unless an architecture suitable for such an iterative algorithm is developed, the decoder should be implemented without pipelining. However all calculations of an iteration are performed in a single clock cycle, maximum clock frequency achieved is not bad.

Synthesis results showed that the bidirectional Fano decoder consumes two times FPGA resources when compared with the unidirectional Fano decoder. It was an expected result. The merging circuit that exists only in the bidirectional Fano decoder is very small, moreover it can be ignored. The bidirectional Fano decoder reduces average decoding delay more than 50%, so we can basicly say that the bidirectional Fano decoder works faster (in terms of throughput per area) than two parallel unidirectional Fano decoders while consuming the same resources. The only bad point of the bidirectional Fano decoder is BER. BER of the bidirectional Fano decoder is higher than BER of the unidirectional Fano decoder.

If the SNR value increases, throughput of the Fano decoder increases dramatically and it also works faster (in terms of throughput per area) than the Viterbi decoder. At low delta values BER is a little bit improved, but the throughput decreases more. Results at delta=8 are more satisfactory.

The Fano decoder consumes too little resources when compared with the Viterbi decoder, but clock frequency of the designed decoder is not faster than the Viterbi's. Despite the clock frequency, throughput per unit area of the Fano decoder is better than the Viterbi decoder at high SNR values such as five and six.

The constraint length was seven in this study. The Fano decoder may become more advantageous at higher constraint lengths. Due to the expectations, the Fano decoder can achieve a better bit error rate with the same resources that the Viterbi decoder consumes by increasing the constraint length. And also number of checked states for merging can be increased to improve BER of the bidirectional Fano decoder.

As a future work, the Fano decoder can be synthesized for ASICs. Then the Fano algorithm achieves a higher throughput. In addition, the unidirectional Fano decoder and the bidirectional Fano decoder can be compared in terms of real area units instead of FPGA resources. The Fano decoder can also be implemented in GPUs. Since GPUs can process more data in parallel, they can achieve a higher throughput than conventional processors.

REFERENCES

Books

Han, Y.S. and Chen, P.-N., 2002. Sequential decoding of convolutional codes. *Encyclopedia of Telecommunications*. John Proakis (Ed.). New York: Wiley, pp. 2140-2164.

Lin, S. and Costello, D.J., Jr., 1983. *Error Control Coding: Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

Periodicals

Benaissa, M. and Yiqun, Zhu, 2007. Reconfigurable hardware architectures for sequential and hybrid decoding. *IEEE Transactions on Circuits and Systems I.* **54** (3), pp. 555-565.

Chen, B. and Sundberg, C.-E.W., 2000. List Viterbi algorithms for wireless systems. *Vehicular Technology Conference Proceedings, Tokyo.* **2**, pp. 1016-1020.

Chevillat, P. and Costello, D., Jr., 1977. A multiple Stack algorithm for erasurefree decoding of convolutional codes. *IEEE Transactions on Communications.* **25** (12), pp. 1460-1470.

Elias, P., 1955. Coding for noisy channels. *IRE Conv. Rec.* **3** (4), pp. 37-46.

Fano, R.M., 1963. A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*. **9** (2), pp. 64-74.

Gorji Zadeh, S.A. and Soleymani, M.R., 2005. An adaptive M-algorithm convolutional decoder. *Vehicular Technology Conference.* **4**, pp. 2177-2181.

Hashimoto, T., 2005. Bounds on a probability for the heavy tailed distribution and the probability of deficient decoding in sequential decoding. *IEEE Transactions on Information Theory*. **51** (3), pp. 990-1002.

Hur, S., Cha, J. and Kang, J., 2006. Fano-based iterative sequential detection algorithm for double-sttd system. *IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications.*

Jelinek, F., 1969. Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development.* **13** (6), pp. 675-685.

Kaiping, Li and Kallel, S., 1999. A bidirectional multiple Stack algorithm. *IEEE Transactions On Communications.* **47** (1), pp. 6-9.

Kaiping, Li and Kallel, S., 1997. Bidirectional sequential decoding. *IEEE Transactions on Information Theory*. **43** (4), pp. 1319-1326.

Kaiping, Li and Kallel, S., 1991. Bidirectional sequential decoding for convolutional codes. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing.* **1**, pp. 200-204.

Katakol, B.S. and Maskara, S.L., 1987. Performance of modified Fano algorithm in AWGN and fading channels. *Proceedings of the IEEE.* **75** (7), pp. 962-964.

Katakol, B.S. and Maskara, S.L., 1985. Computationally efficient modified Fano algorithm for sequential decoding. *Electronics Letters.* **21** (23), pp. 1109-1111.

Gorji Zadeh, S.A. and Soleymani, M.R., 2005. An adaptive M-algorithm convolutional decoder. *Vehicular Technology Conference.* **4**, pp. 2177-2181.

Lee, J., 1998. Richard Wesley Hamming: 1915-1998. *IEEE Annals of the History of Computing.* **20** (2), pp. 60-62.

Lin, Y. and Tu, S.H., 1997. Modified multiple Stack algorithm for decoding convolutional codes. *Communications, IEE Proceedings.* **144** (4), pp. 221-228.

Long, E.M. and Bush, A.M., 1989. Decision-aided sequential intersymbol interference sequence estimation for channels. *IEEE International Conference on Communications.* **2**, pp. 841-845.

McClure, J.H. and Joiner, L.L., 2001. Soft decision decoding of Reed-Solomon codes using the Fano sequential algorithm. *Proceedings of the IEEE, SoutheastCon 2001*. pp. 131-135.

Mohammad, M., Je-Hong, J., Ravishankar, C. and Barnett, C., 2008. A comparison between the M-algorithm and the list Viterbi algorithm. *Military Communications Conference, IEEE.* pp 1-5.

Muhammad, K. and Letaief, K.B., 1992. Statistical evaluation of the error rate of the Fano and stack algorithm decoders. *Communications on the Move, Singapore ICCS/ISITA '92.* **2**, pp. 872-876.

Ozdag, R.O. and Beerel, P.A., 2006. An asynchronous low-power high-performance sequential decoder implemented with QDI templates. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.* **14** (9), pp. 975-985.

Pan, W. and Ortega, A., 2001. Buffer control for variable complexity Fano decoders. *Global Telecommunications Conference, IEEE.* **1**, pp. 176-180.

Pottie, G.J. and Taylor, D.P., 1989. A comparison of reduced complexity decoding algorithms for trellis codes. *IEEE Journal on Selected Areas in Communications.* **7** (9), pp. 1369-1380.

Pun, P. and Ho, P., 2005. Bit error probability and computational complexity of bi-directonal fano multiple symbol differential detectors. *International Conference on Information, Communications and Signal Processing.* pp. 1540-1545.

Ran, Xu, Kocak, T., Woodward, G., Morris, K. and Dolwin, C., 2011. High Throughput Parallel Fano Decoding. *IEEE Transactions On Communications.* **59** (9), pp. 2394-2405.

Ran, Xu, Kocak, T., Woodward, G., Morris, K. and Dolwin, C., 2009. Bidirectional Fano algorithm for high throughput sequential decoding. *IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications.* pp. 1809-1813.

Senk, V. and Radivojac, P., 1997. The bidirectional Stack algorithm. *IEEE International Symposium on Information Theory.*

Viterbi, A.J., 1967. Error bounds for convolutional coding and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory.* **13** (2), pp. 260-269.

Wozencraft, J.M., 1957. Sequential decoding for reliable communication. *IRE Nat. Conv. Rec.* **5** (2), pp. 11-25.

Zhou, T., Chen, D., Xu, M. and Yu, J., 2006. Constrained list Viterbi algorithm and its application in image transmission via noisy channels. *IET International Conference on Wireless, Mobile and Multimedia Networks.* pp 1-4.

Zigangirov, K.Sh., 1966. Some sequential decoding procedures. *Probl. Peredachi Inf.* **2** (4), pp 13-25.

Other Publications

Forney, G.D., Jr., (1967). Coding system design for advanced solar missions. *Technical report.* California: NASA Ames Research Center.

Han, Y.S., Sequential decoding of binary convolutional codes [online], Puli, National Chi Nan University. http://web.ntpu.edu.tw/~yshan/Sequential_decoding_convolutional_codes.pdf [accessed 22 August 2012].

Xilinx, Inc., 2011. LogiCORE IP, Viterbi Decoder v7.0, Product Specification [online], San Jose, CA: Xilinx Inc. http://www.xilinx.com/support/documentation/ip_documentation/viterbi_ds247.pdf [accessed 17 October 2011].

CURRICULUM VITAE

Full Name : Ahmet Kakacak

Adress : Gülbahar Mah, Cemal Sururi Sk, 35/3, Şişli/İSTANBUL

Birth Place / Date : Eskişehir / 07.05.1987

Foreign Languages : English (Advanced)

High School : Eskişehir Anatolian High School, 2005

B. Sc : Electrical & Electronics Engineering, Yeditepe University, 2009

M. Sc : Electrical & Electronics Engineering, Bahçeşehir University, 2012

Name of Institute : Natural And Applied Sciences

Name of Program : Electrical & Electronics Engineering

Work Experience :

October 2010 - Present

Vestek R&D Corp., Istanbul, Full-time Digital Design Engineer at Vestek R&D Corp.