Convolutional codes are widely adopted in wireless communication systems for forward error correction. Creonic offers you an open source Viterbi decoder with AXI4-Stream interface, that is capable of decoding most of the convolutional codes as defined by various standards.

Key features of the core are:

- Design-time configuration of encoder polynomials (different constraint lengths and different code rates).
- Support for recursive and non-recursive convolutional codes.
- Windowing technique for reduced latency and memory requirements (with acquisition).
- Design-time configuration of quantization, maximum window size, RAM usage (distributed RAM vs. Block RAM).
- Run-time configuration of block length.
- Run-time configuration of window length and acquisition length.
- Block-to-block on-the-fly configuration.

Your benefits are:

- Configurable for most standards that apply convolutional codes (GSM, UMTS, CDMA, CDMA2000, WiMAX, WiFi, DAB, ...).
- Pipelined design for high payload throughputs (about 1 bit per clock cycle).
- AXI4-Stream interface for simple integration.
- Up to 250 MHz on Xilinx Virtex-6 FPGA (Speedgrade 1).
- VHDL source code available under GPL license (hosted at opencores.org).
- Commercial support and licenses available.

Contents

1 Block Description

1.1 Block Diagram

Figure 1.1 describes the interfaces of the Viterbi decoder, as well as its internal building block structure.

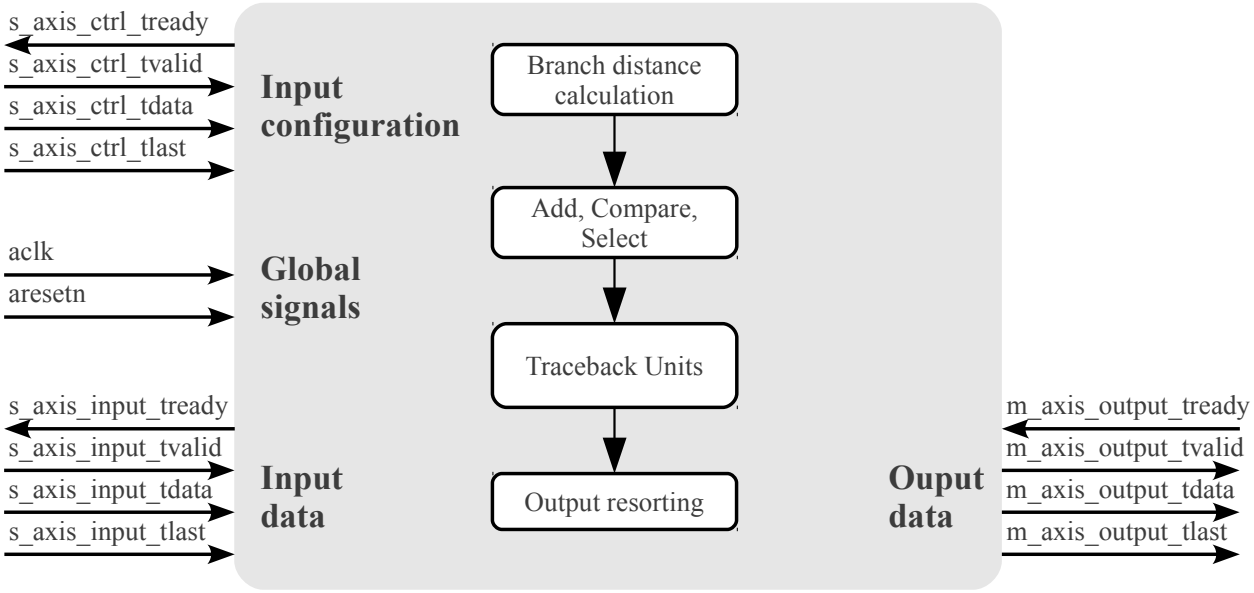

Figure 1.1: Block diagram of the Viterbi decoder

The core uses three AXI4-Stream interfaces (data input, data output, and control), see Section 1.3 for more information. The core allows a simple adaptation for different standards, as well as different requirements. Section 1.2 shows a list of parameters that can be modified at design-time.

1.2 Generic Parameters

Table 1.1 shows a list of parameters that can be adjusted before synthesis of the IP core.

Parameter	Type	Description
NUMBER_PARITY_BITS	natural	Number of parity bits, specifying the code rate.
PARITY_POLYNOMIALS	natural array	Parity polynomials, describing the convolutional code.
FEEDBACK_POLYNOMIAL	natural	If any, the polynomial for recursion is given here.
BW_LLR_INPUT	natural	Bit width of the signed input log-likelihood ratios (LLRs).
MAX_WINDOW_LENGTH	natural	The maximum size of a window and the acquisition.
DISTRIBUTED_RAM	boolean	Using distributed or block RAM.

Table 1.1: Generic parameters defined in */packages/pkg_param.vhd*

Changing these parameters affects area and error correction capabilities of the design.

NUMBER_PARITY_BITS

The number of used parity bits has to correspond to the used PARITY_POLYNOMIALS and describe the code rate. Common values are 2 ($R = 1/2$), 3 ($R = 1/3$), and 4 ($R = 1/4$). Even though lower rates are not applied in the standards, it can be supported by the design.

PARITY_POLYNOMIALS

The parity polynomials are usually defined by a standard. The convolutional code polynomials values have to be given in a decimal representation. The most significant bit is the leftmost tap in the convolutional code $g(D) = D^n + ... + D^8 + D^4 + D^2 + 1$. Polynomials are given in an array of naturals. An overview of convolutional standard code polynomials in octal notation is given in Table 1.2. For configuration of the core, please us decimal notation.

Polynomials	States	Rate R	g_0	g_1	g_2	g_3
GSM/EDGE	16	$1/2$	33_8	23_8		
	16	$1/3$	33_8	25_8	37_8	
IEEE 802.15.3c, WiMax, DVB-H, DVB-S, DVB-T	64	$1/3$	133_8	171_8	165_8	
DAB	64	$1/4$	133_8	171_8	145_8	133_8
IEEE 802.11a/b/g/n, GSM	64	$1/2$	133_8	171_8		
	64	$1/3$	133_8	171_8	145_8	
CDMA 2000, UMTS	256	$1/2$	753_8	561_8		
	256	$1/3$	557_8	663_8	561_8	
	256	$1/4$	765_8	671_8	513_8	473_8

Table 1.2: Polynomials in octal notation

FEEDBACK_POLYNOMIAL

If a recursive convolutional code is used, the feedback polynomial is given here. Representation is like PARITY_POLYNOMIALS. If no recursion is used, the value has to be set to 0.

BW_LLR_INPUT

The bit width of the input LLRs is usually set to 4 for a sufficient error correction capability. Since LLR values are signed values, a value of 4 will result in a 3 bit magnitude. There is improvement above 8 bit per LLR value this is the maximum bit width in the current design.

MAX_WINDOW_LENGTH

The decoder is able to reconfigure the window and acquisition length at runtime. However since there is no boundary in general, it is necessary to define a maximum window length at design-time. This limits the amount of RAM used by the decoder.

DISTRIBUTED_RAM

Set to **true** if distributed RAM shall be used (Xilinx). Set to **false** if block RAM shall be used (Xilinx, Altera).

1.3 Interface Description

The core uses the AXI4-Stream interfaces for a simple integration.

1.3.1 AXI4-Stream Signals

The AXI4-Stream interface is an interface for point to point connections, connecting one master (data source) with on slave (data drain). The signals defined in the AXI4-Stream interface use prefixes to define the master/slave and suffixes to define the meaning of the signals. The prefix "m_axis_" indicates that this is a AXI4-Stream master, while the prefix "s_axis_" indicates that the signals belong to an AXI4-Stream slave. The signal suffixes are given in Table 1.3. Section 2.1 depicts the timing diagram of a typical AXI4-Stream

Suffix	Width (bits)	Description
_tvalid	1	The master asserts tvalid if tdata (and tuser) is valid.
_tdata	n	tdata delivers the payload data stream.
_tlast	1	Indicates that this is the last data transfer of the current block.
_tuser	n	Side channel information, only used where applicable.
_tready	1	Signal is asserted by the slave if it is ready to receive data.

Table 1.3: Configuration signals for the IP core

connection.

1.3.2 General Signals

Table 1.4 lists general signals required by the core. These signals apply to all AXI4-Stream interfaces used. Information about the clock speed is given in Chapter 3.

Pin	Sense	Width (bits)	Description
aclk	in	1	Clock, all synchronous operations within the core happen to the rising edge of aclk.
aresetn	in	1	Reset is synchronous and active low.

Table 1.4: Global signals for the Viterbi decoder

1.3.3 Configuration Interface

Configuration of the core is done using the AXI4-Stream interface. Since the core supports a block-to-block on-the-fly configuration, a new configuration has to be performed for each block. Only after configuration, the core will indicate that it is ready to read input data. Table 1.5 lists the configuration signals.

Pin	Sense	Width (bits)	Description
s_axis_ctrl_tvalid	in	1	Indicates the data at s_axis_ctrl_tdata is valid.
s_axis_ctrl_tdata	in	32	Window and acquisition length is delivered
s_axis_ctrl_tlast	in	1	Indicates the last configuration bit of s_axis_ctrl_tdata.
s_axis_ctrl_tready	out	1	Signal is set to '1' if the decoder is ready to receive data.

Table 1.5: Configuration signals for the IP core

s_axis_ctrl_tdata

The unsigned window length is placed in the upper 16 bit of the signal. The unsigned acquisition length is placed in the lower 16 bit of the signal.

1.3.4 Data Input/Output Interface

Table 1.6 shows the data interface signals of the core. In general the LLR values are given from a demapper. It is not necessary, the demapper has an AXI interface, but some controller has to handle the handshake protocol, which is described in Chapter 2.

Pin	Sense	Width (bits)	Description
s_axis_input_tvalid	in	1	Valid signal for parity data
s_axis_input_tdata	in	32	Parity LLR values
s_axis_input_tlast	in	1	Indicates the last LLR value parity set
s_axis_input_tready	out	1	Ready if configuration is done
m_axis_output_tvalid	out	1	Valid signal for decoded bit stream
m_axis_output_tdata	out	1	Decoded bit stream
m_axis_output_tlast	out	1	Indicates the last bit decoded bit of a block
m_axis_output_tready	in	1	The environment signals it is ready to receive data

Table 1.6: Data input and output signals for the IP core

s_axis_input_tdata

The soft input LLR values have to be given in signed two's complement representation as shown in Table 1.7. The LLR values are given in four blocks of eight bits. A block of eight bits has to contain one signed LLR value, while the number of used bits corresponds with the parameter setup (BW_LLR_INPUT). Depending on the number of parity polynomials the unused bytes will be ignored. Table 1.8 shows an example of a valid input. Using a 32-bit word is beneficial when the decoder is used in combination with a CPU such as the MicroBlaze from Xilinx.

Probability	two's complement (2 downto 0)	decimal
strongest 1	100	-4
	101	-3
	110	-2
	111	-1
	000	0
	001	1
	010	2
strongest 0	011	3

Table 1.7: Signed representation of soft input LLR for BW_LLR_INPUT = 3.

Parity	3	2	1	0
s_axis_intput_tdata (31 downto 0)	0000 0000	0000 0110	0000 0011	0000 0100
LLR meaning	not present	-2	3	-4
represented bit	not present	1	0	1

Table 1.8: Input data format for Rate = $1/3$, NUMBER_PARITY_BITS = 3, BW_LLR_INPUT = 3.

2 Functional Description

This chapter explains details about the scheduling behavior of the decoder. If the core is used in a chain the timing has to be cosidered. Therefore the external timing behavior is given here.

2.1 General AXI4-Stream Handshake

The AXI4-Stream interface is based on the ARM standard definition [1]. Only when both, tvalid by the master and tready by the slave, are asserted, data is transfered from master to slave. The master has to assert the tvalid signal if data is available and is not allowed to wait for the slave to assert tready beforehand. Once the master has asserted tvalid it has to remain asserted until tready from the slave is asserted. The slave in contrast is allowed to deassert tready at any time and wait for the master's tvalid assertion. Once the last data information of the stream arrives, the master asserts tlast. The optional tuser signal has to be stable when tvalid is asserted. The representation of data within tdata and tuser can be customized according to the current requirements. Figure 2.1 shows an example handshake.

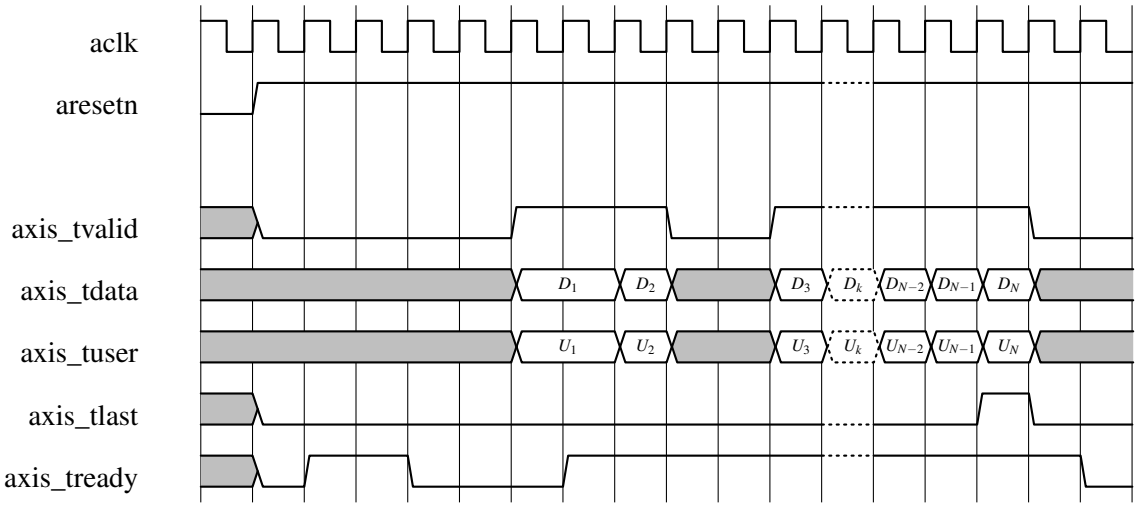

Figure 2.1: Example of an AXI4-Stream handshake

2.2 Decoder Interface Timing Example

For using the decoder in a receiver chain the external timing is needed. In Figure 2.2 decoding of a common block is presented. At first the complete design is reset and after this the decoder is configured. Once it is configured, the decoder is ready to receive data. Depending on the configuration the decoder starts to output decoded bits some time later. The latency roughly can be estimated by $2 \cdot window_length$. The succeeding chain component has to be ready too. Unless the decoder is interrupted by missing input or an low ready signal from another component, the decoder outputs one decoded bit per cycle. When the last LLR value arrives the previous component has to wait fore some time, until the decoder is ready again to accept a

new configuration and data. The decoder sets the last signal, when the last decoded bit is available to the following component.

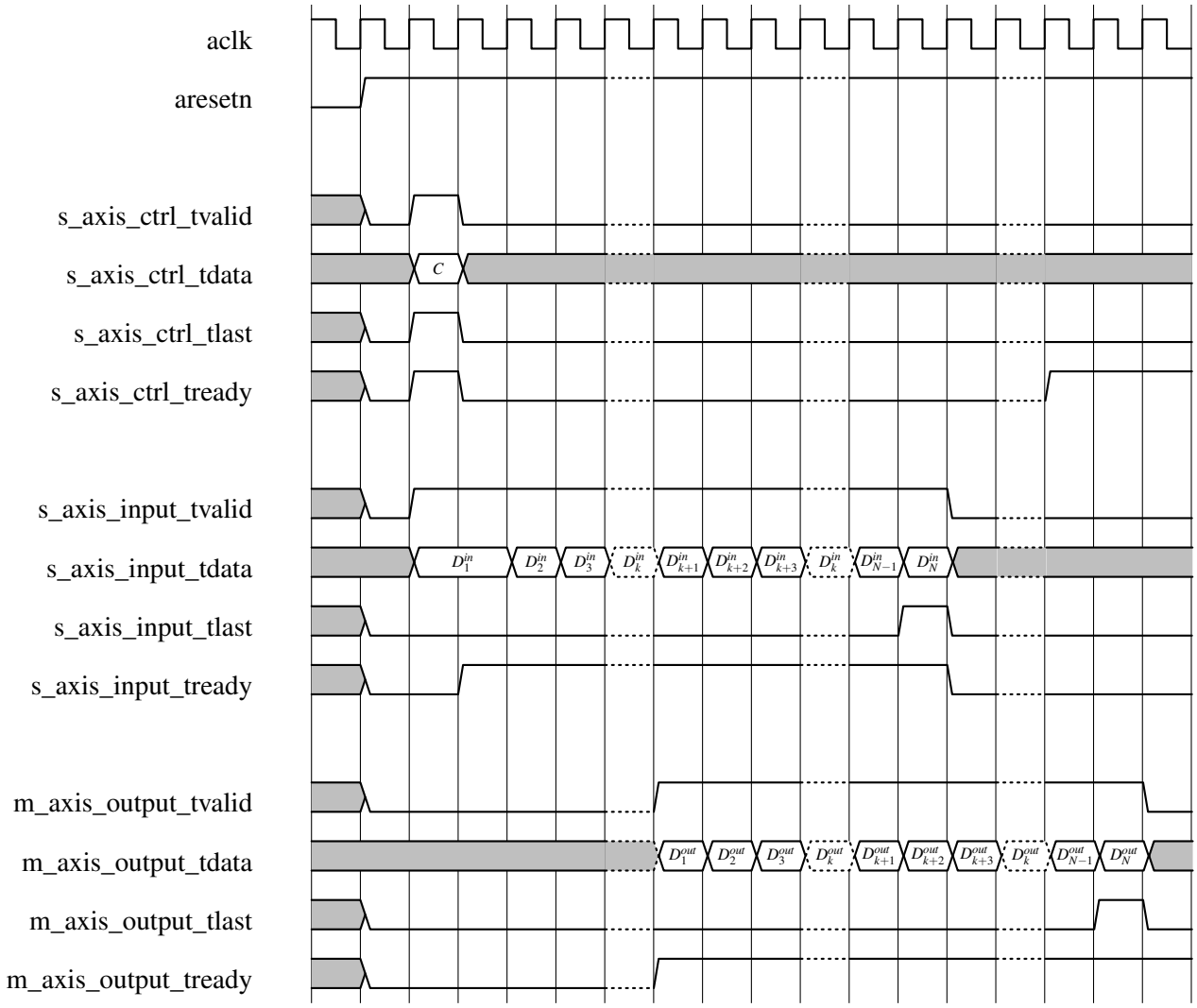

Figure 2.2: Decoder timing for configuration, data input and output

Configuration

Because configuration is done in one cycle s_axis_ctrl_tlast is not necessary to set all the time. Therefore it is an optional signal. However it is possible to reconfigure the decoder for each arriving block, this is no desired most of the time. For the decoder it is necessary to read the configuration before decoding a new block, but it is allowed to set s_axis_ctrl_tvalid to high and s_axis_ctrl_tvalid to a constant value. An example for a fixed decoder configuration is shown in Figure 2.3.

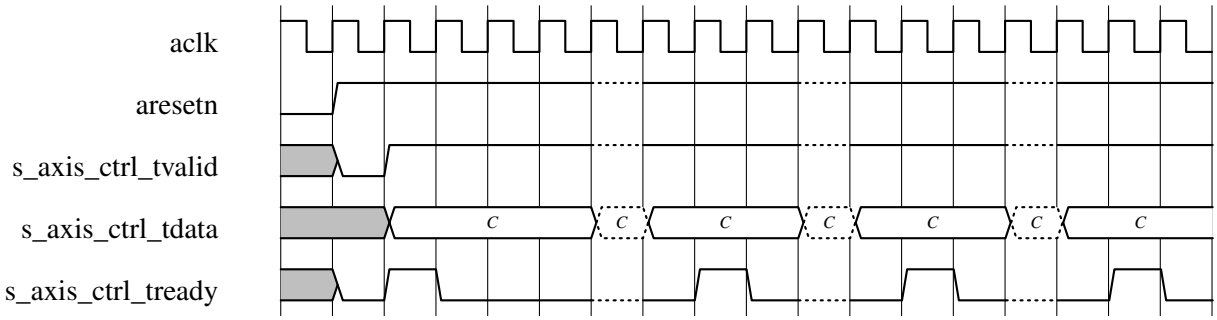

Figure 2.3: Fixed decoder configuration

2.3 Latency

The latency of the design is defined by the time from the first input bit arrives until the first output bit is available. Here the definition assumes there is one output bit at each cycle, which is the true for the implementation. The latency of the Viterbi decoder can be calculated by:

$$latency_{dec} = 2 \times (window\_length + acquisition\_length) + 6 \tag{2.1}$$

3 Implementation Results

The following results were obtained using normal effort for synthesis, placement, and routing. Better results might be obtained by tweaking the tool options. Please note that variations in the results are possible, depending on tool versions, tool parameters, constraints, FPGA usage, etc.

3.1 Xilinx

3.1.1 Virtex-6

FPGA	MAX_WINDOW_LENGTH / BW_LLR_INPUT	Constraint length / DISTRIBUTED_RAM	LUTs	FFs	BRAMs (36k)	Frequency (MHz)
xc6vlx240t-1	96 / 4	7 / false	2984	1302	4	250
xc6vlx240t-1	96 / 5	7 / false	3491	1434	4	250
xc6vlx240t-1	96 / 4	7 / true	3481	1565	0	244
xc6vlx240t-1	96 / 4	9 / false	12315	5314	16	200
xc6vlx240t-1	48 / 4	7 / false	2875	1203	4	244

Table 3.1: Resource utilization for Virtex-6, ISE 13.3.

3.1.2 Spartan-6

FPGA	MAX_WINDOW_LENGTH / BW_LLR_INPUT	Constraint length / DISTRIBUTED_RAM	LUTs	FFs	BRAMs (18k)	Frequency (MHz)
xc6slx45-2	96 / 4	7 / false	3054	1309	8	142
xc6slx45-2	96 / 5	7 / false	3536	1441	8	142
xc6slx45-2	96 / 4	7 / true	4068	1693	0	136
xc6slx45-2	96 / 4	9 / false	12819	4972	32	100
xc6slx45-2	48 / 4	7 / false	3339	1320	8	142

Table 3.2: Resource utilization for Spartan-6, ISE 13.3.

3.2 Altera

3.2.1 Cyclone IV

FPGA	MAX_WINDOW_LENGTH / BW_LLR_INPUT	Constraint length / DISTRIBUTED_RAM	LE	FFs	BRAMs M9K	Frequency (MHz)
ep4cgx30cf23c6	96 / 4	7 / false	4124	2012	8	192
ep4cgx30cf23c6	96 / 5	7 / false	4399	2080	8	180
ep4cgx30cf23c6	96 / 4	9 / false	16035	7136	32	153
ep4cgx30cf23c6	48 / 4	7 / false	3996	1890	8	187

Table 3.3: Resource utilization for Cyclone-IV, Quartus II 11.0.

4 Communications Performance

Many communication standards define convolutional codes for FEC. There are several parameters, like polynomials, frame size, input bit quantization and window length, which can be adapted depending on the application. Mobile speech communications do not need a high bandwidth, but a low power consumption for saving battery and a good error correction because of the rough environment. On the other hand, wired connections need low latency and high bandwidth.

Therefore it is necessary to know how the parameters influence the behavior of the code. Standards try to define a solution for a specific application, but do not describe all parameters. Mostly the code polynomials and the frame length are given, while window length and input bit length have to be chosen by the designer.

4.1 Polynomials

Communication standards define the code polynomials and information frame length.

As already mentioned the convolutional codes depend on the application, but also differ depending on the release time. Old standards like original GSM from 1990 are using code polynomials with a small constraint length to keep them simple, while simple means fast computation, small chip size and low energy consumption.

Newer standards like CDMA 2000 consider usage of more recent chip technologies. Chip size and computation speed now are handled at the technology level. Therefore it is possible to increase the constraint length. Furthermore the physical transmission bandwidth increased, which rises the number of errors on the channel. Here it is necessary to focus on a better error correction ability, which is realized by a higher constraint length and number of parity bits per information symbol.

To sum up, a standard defines the parity polynomials as well as the length of an information frame. Independently of implementation it is possible to compare different code polynomial sets and the resulting error correction ability.

Different Code Polynomials

Choosing code polynomials is a complex task, trying to optimize the error correction ability. For a given constraint length this optimization problem mostly leads to a single result, which is used by all standards. Using non optimum polynomials in general results in a much worse error correction ability. In order to confirm this statement a standard polynomial set shall be compared to others. The results are presented in Figure 4.1. For simplicity of computation parity polynomials $[33, 23]$ from the GSM are selected as a reference. In order to be generic the bit error rate is compared here, so the frame length does not matter, but for the sake of completeness is set to 228, as described in GSM.

The reference polynomials constraint length is 5 and the rate is $1/2$. For comparison a code with the same constraint length and rate is selected. The polynomials are selected randomly, but do not result in catastrophic or malicious code.

First the code $[24, 21]$ is simulated in Figure 4.1, which obviously results in a worse error correction ability to the reference. The second code is $[36, 27]$, which has a higher number of taps as the reference. But again it is a lot worse than the GSM, so there is no relation from tap number to error correction ability. At last a code with a higher constraint length is used, because higher constraint length increases the memory

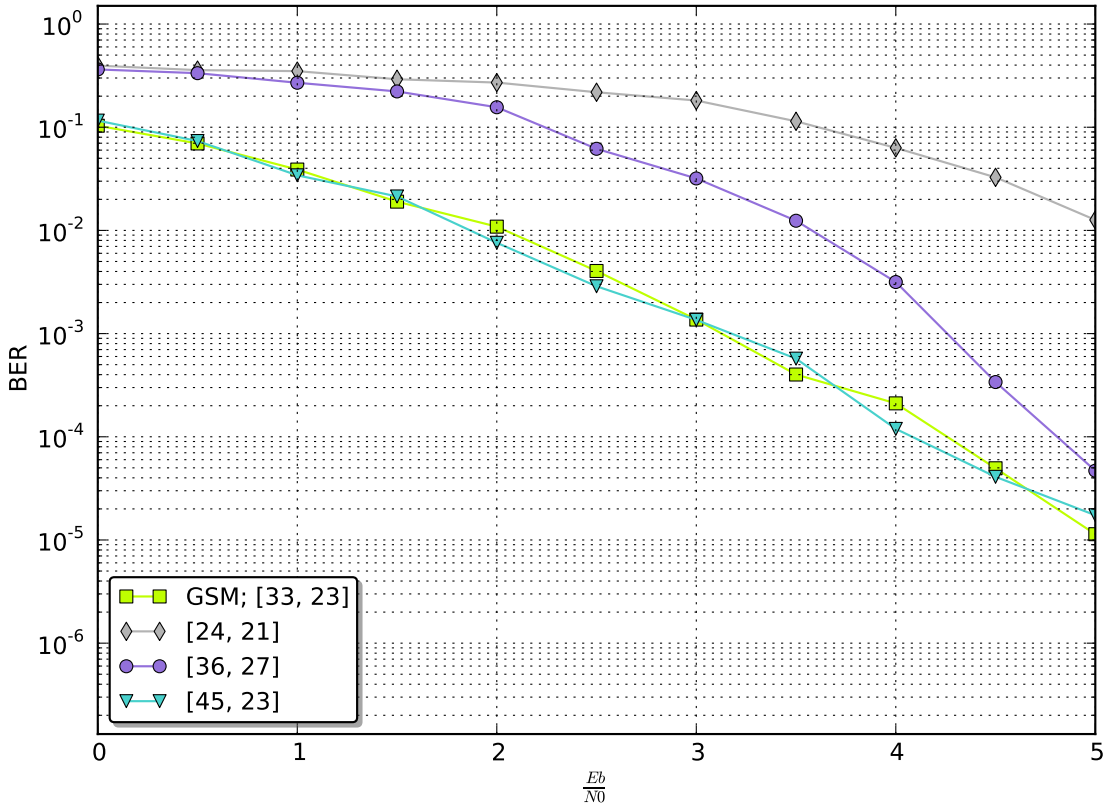

Figure 4.1: Variable polynomials; Rate = 1/2

depth and the information contained in a parity bit. The selected code $[45,27]$ now performs similar to the reference, but at the cost of $K = 6$ instead of $K = 5$.

To sum up, it is shown that the code polynomials have a big influence in the error correction ability. Even choosing a higher constraint length does not necessarily surpass a well chosen code in terms of error correction, while there is no effect on size, latency or speed. Therefore the code polynomial sets used in different standards, in general is the same or similar when the same constraint length and rate is used.

For example the common rate $1/3$ with constraint length 7 uses $[133,171,165]$ code polynomials for WiMAX, DVB-H, DVB-S, DVB-T and IEEE 802.15.3c. Only GSM defines different but similar $[133,171,145]$ code polynomials for this rate. In order to provide different levels of error correction, for example CDMA 2000 defines different code rates. GSM on the other hand additionally defines different constraint length as well.

Different Rates

First CDMA 2000 is considered here for the correction behavior of different rates. From theory the lower rate must achieve a better error correction, since there is more redundancy. But with a lower rate more data has to be transmitted for same information, as well as the amount of calculations increases. Now the rates $1/4$, $1/3$ and $1/2$ with a fixed constraint length 9 are considered. The compared polynomials are shown in Table 4.1 and visualized in Figure 4.2. Again the bit error rate is compared here and it can be senn, that the correction behavior is as expected from theory. A higher rate increases the correction ability. But the more interesting fact is, the gain from $1/2 \rightarrow 1/3$ and $1/3 \rightarrow 1/4$. From $1/2$ to $1/3$ the gain is about 0.5dB, while the gain from $1/3$ to $1/4$ is about 1.65dB. It is getting less with each step. But while the hardware

Rate	g_3	g_2	g_1	g_0
1/2			735	561
1/3		557	663	711
1/4	765	671	513	473

Table 4.1: CDMA 2000 convolutional code polynomials

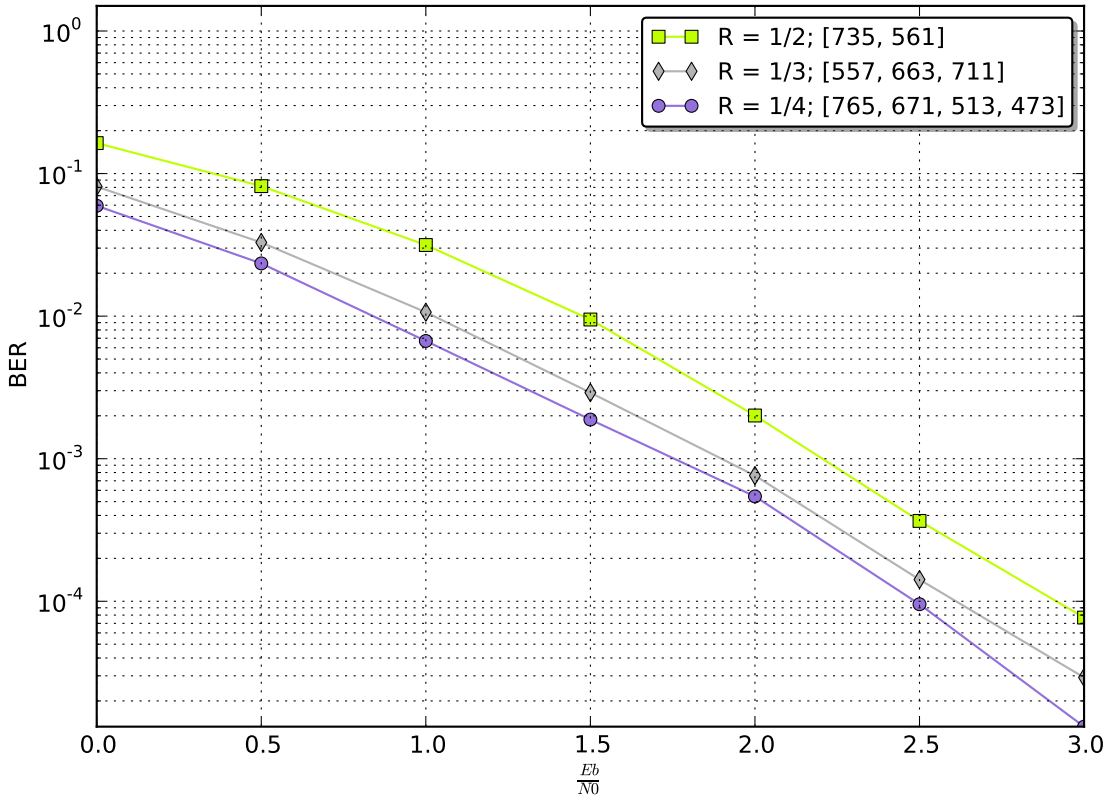

Figure 4.2: Variable code rate; Constraint length = 9

increases only slightly, the amount of sent data increases a lot. Therefore using low rates is a good choice if the channel is likely to have many errors, which in some way is a result of the need to transmit much data. It is a trade off between correction gain and cost for transmission.

Different Constraint Lengths

Another way to improve error correction ability is using a larger constraint length K. Theory again says a larger constraint length increases code performance. This is because the information contained in a parity bit is larger, but also because there is a greater freedom of choosing the polynomials. In this case the amount of transmitted data stays the same. For this the polynomials in Table 4.2 are considered. Again the result is similar to the behavior of altering the rates. Instead of growing the amount of transmitted data the amount of hardware increases a lot, since the number of states is calculated by 2^{K-1}. Historically the constraint length grew more than the rate has been decreased, because new technology is able to handle the additional cost of hardware.

Appearance	Constraint length	g_1	g_0
GSM	5	33	23
GSM	7	171	133
UMTS	9	753	561

Table 4.2: codes with fixed rate, but different constraint length

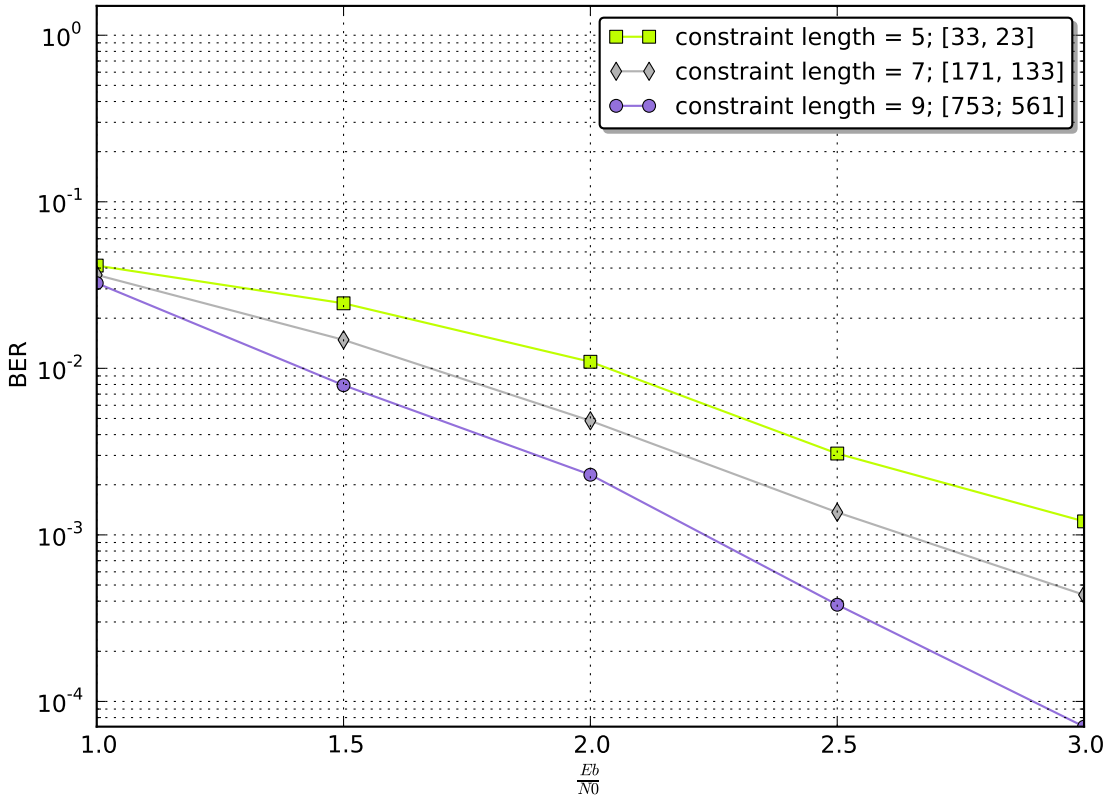

Figure 4.3: Variable constraint length; Rate = 1/2

Compare Rate and Constraint Length

It leaves the question which parameter, rate or constraint length, has the higher influence. This helps to compare different standards. For comparison the DVB-T code $[133,171,165]$ is compared with the CDMA 2000 code $[753,561]$ As Figure 4.4 shows a lower rate has a better error correction even with a lower constraint length. This is not surprising, since a larger constraint length only contains some more information about the past. With a lower rate on the other hand there is a additional information frame transmitted. But there are reasons to use a larger constraint length. Standards like CDMA 2000 or UMTS are using large constraint lengths, since the used technology is able to provide the amount of hardware, while increasing the rate results in a worse usage of the available bandwidth. In case a larger constraint length is able to correct sufficiently many data frames of a stream there is no need to use a lower rate and therefore bandwidth.

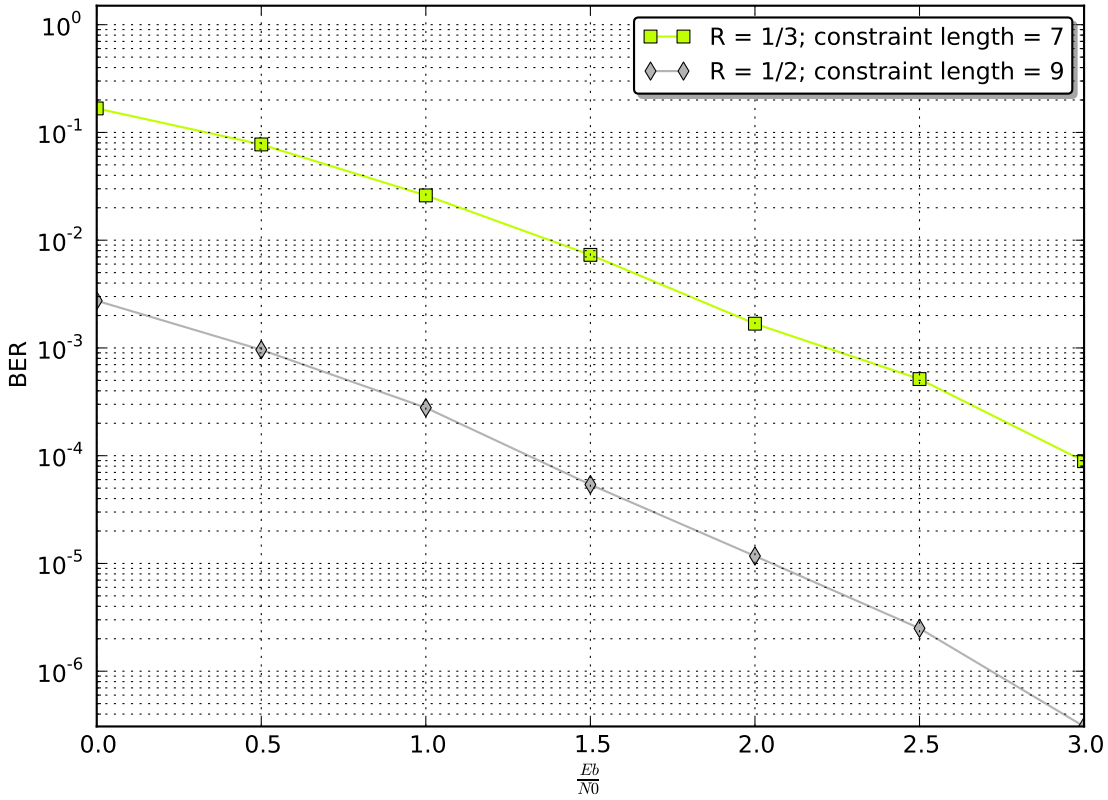

Figure 4.4: Comparison of rate and constraint length influence

4.2 Frame Size

Information data is sent in frames with a given size. Like code polynomials the frame size is defined in the standards and varies for different types of packets. Synchronization packets for example are mostly smaller than packets containing information for the user. On the one hand synchronization packets do not need to be large, because they do not contain much information, while on the other hand small frames produce a lot of overhead. Overhead is produced by the packets on a higher transmission level, but also due to tailing and stalling transmission. This is why user information packets are larger since there is a much data to transmit and an decreased overhead saves bandwidth.

Either way the frames have to be transmitted correctly. A frame is incorrect if there is at least one decoded bit, that differs from the original source bit. It does not matter how many bit errors happen within a frame. This way the frame error rate "FER" shows how many frames where incorrect. Since there are different sized frames it is important to know the FER in order to decide which size is reasonable for a given application and packet type.

To show the behavior of different frame sizes, again the GSM standard is considered. Here a code rate $1/2$ with polynomials 133 and 171 is chosen, since these parameters are also used for Wifi and DVB. The frame size for these parameters starts at 20 and is followed by any natural number up to 870. In order to show the behavior the FER for some frame sizes is shown in Figure 4.5. As already discussed in Section 4.1 the bit error rate is not affected by the frame size.

For FER it can be observed, that an increasing frame size increases the FER. An increasing frame size results in an inceasing number of bits per frame. This way a larger frame has a higher probability a bit error

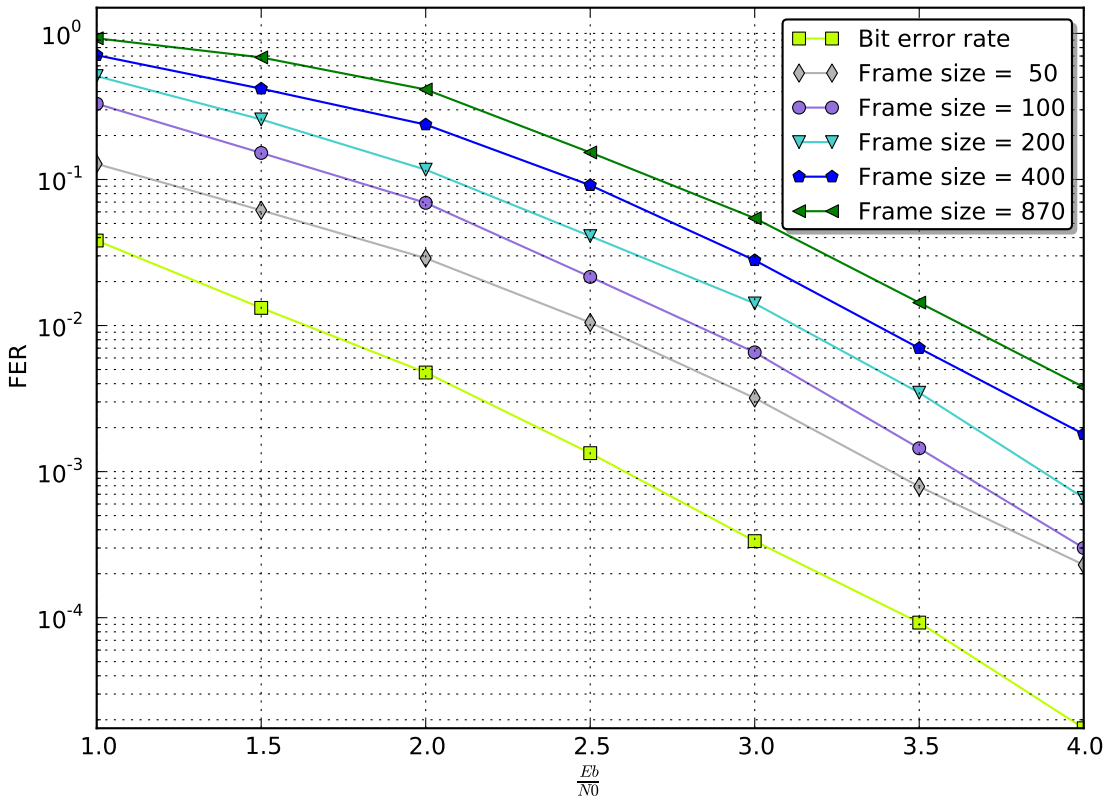

Figure 4.5: Variable frame size; Rate = 1/2; Constraint length = 7

appears. In a simplified example at a bit error rate of 0.5% and 100 bits frames on average only every second frame has an error. But when the frame size is increased to 200 bits per frame, on average every frame has an error.

To show this even more for a few fixed SNR points the FER and the bit error rate of the simulated frame sizes are shown in Table 4.3. Here it is easy to see, that the FER distance between different frame sizes does

	Frame size				
$\frac{Eb}{N0}$	50	100	200	400	870
	Frame error rate				
2	$2.9 \cdot 10^{-2}$	$6.9 \cdot 10^{-2}$	$1.2 \cdot 10^{-1}$	$2.4 \cdot 10^{-1}$	$4.1 \cdot 10^{-1}$
3	$3.2 \cdot 10^{-3}$	$6.6 \cdot 10^{-3}$	$1.4 \cdot 10^{-2}$	$2.8 \cdot 10^{-2}$	$5.4 \cdot 10^{-2}$

Table 4.3: Variable frame size; Fixed SNR Eb/N0 values; Rate = 1/2; Constraint length = 7

not change with an increasing SNR. This can be easily explained by stating that multiplication is a linear function. If bit error rate is multiplied with frame size this also results in a linear relationship, when varying the SNR.

To conclude, a larger frame size has a higher FER, while the FER distance for different frame sizes is independent of the SNR. This way one can say a smaller frame size is more likely to be decoded correctly, but at the cost of more overhead. On the other hand for larger frames the overhead is reduced, but it is more likely to be decoded wrong.

Although the standards define a range of frame sizes, not all are used. Many decoders use the windowing technique, where it is a good choice if the frame length can be divided by the window length. This way the choice of window length may affect the frame length decision.

4.3 Traceback, Window and Acquisition Length

Using larger frames for data transmission reduces the overhead, described Section 4.2. The latency increases as frames get larger. It is claimed, that the acquisition length shall be at least six times of the constraint length. In order to discard as few as possible traceback bits, the acquisition length only shall be as long as necessary.

It is necessary, that there is asymptotically no difference between the non windowed and the windowed version. Therefore a non windowed simulation is used as reference to prove an acquisition length of six times the constraint length is sufficient and necessary. The number of states is increased by the constraint length, so different constraint length shall be observed. Since they are widely used, convolutional codes with constraint length 7 and 9 are chosen. Furthermore the code rate does not matter for the simulation, because the correct path through the trellis is not influenced by the rate. For the same reason the polynomials do not matter and are chosen according to the standards. In the end the length of the frame has to be selected. It again does not matter, but it is a good choice to select a large frame, to have a wide range of different paths. The used parameters are shown in Table 4.4 and the resulting simulations in Figure 4.6 and Figure 4.7.

Fixed code parameters			Acquisition lengths			
Constraint length	Polynomial set	Frame size				
7	$[133, 171, 165]$	800	21	28	35	42
9	$[557, 663, 711]$	800	27	36	45	54

Table 4.4: Variable acquisition length parametes

First it is easy to see, that any acquisition length up to five times the constraint length does not match the reference. But as already stated, when the length is chosen $6K$ the plot fits well to the reference asymptotically, while it does not matter if the constraint length is 7 or 9. The formula $6K$ is always true independent of the constraint length. This can be explained by the fact, that every state is reachable at any time in at most K steps. Therefore the chance to reach the correct path scales linearly with the constraint length.

When the channel error rate is low, even other acquisition lengths get close to the reference, since the way to the correct path is short. To sum up, when windowing is used the acquisition length $6K$ can be chosen safely.

When using windowing the acquisition length is used to find the correct path. For simulation and minimum latency the window length has been chosen equally to the acquisition length. The window length does not have any affect to the error correction ability. A larger window only reduces the number of calculations, at the cost of an increased the latency. In conclusion the windowing technique can be used as an optimization without any influence on the error correction ability.

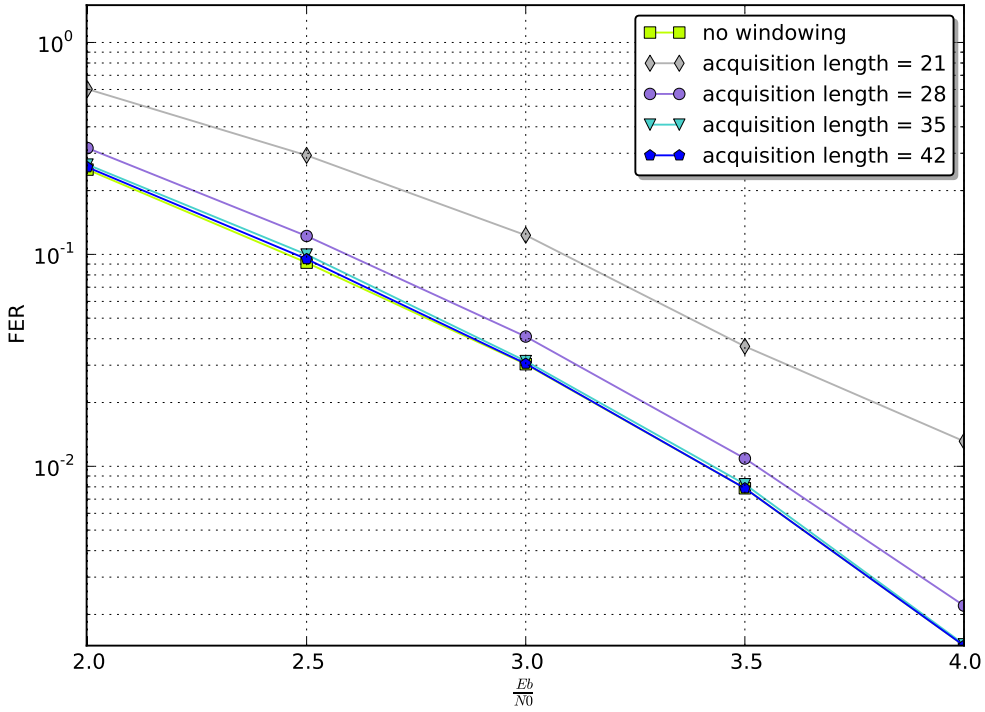

Figure 4.6: Variable acquisition length; Rate = 1/3; Constraint length = 7; Frame size = 800

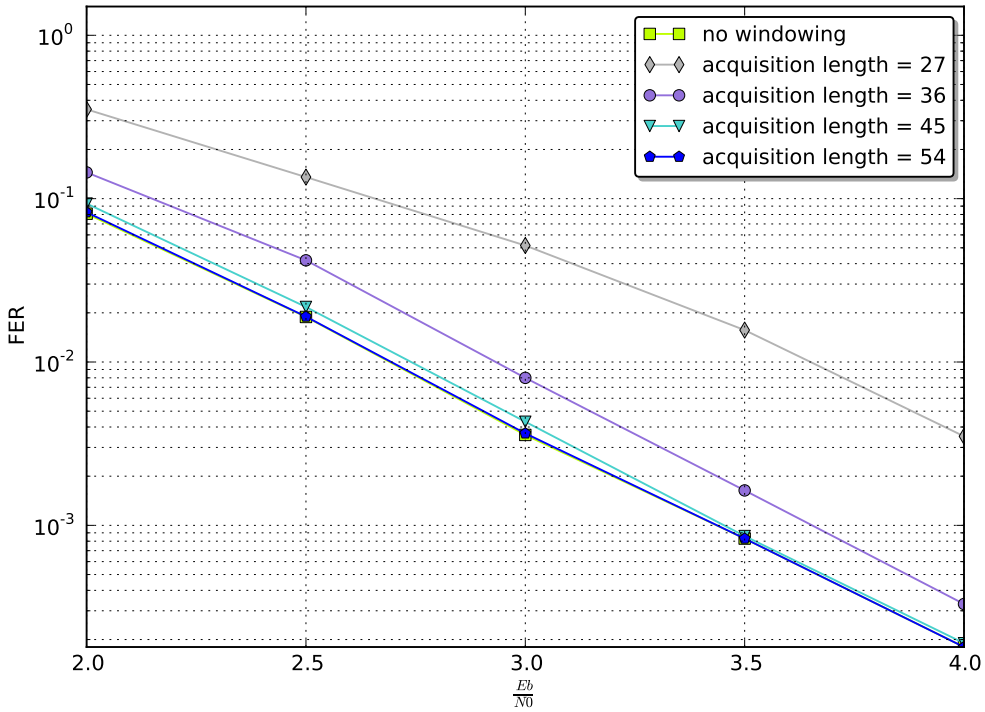

Figure 4.7: Variable acquisition length; Rate = 1/3; Constraint length = 9; Frame size = 800

4.4 Input Bit Quantization

It is discussed in Section 4.1 that the error correction ability depends on the selected polynomials, as well as code rate and the frame length. This parameters affect the convolutional code itself and therefore the behavior of the encoder and decoder. Further, demapping adds a quantization error to the input stream of the decoder. The amount of the quantization error depends on the number of bits the demapper uses to express the physical value. Using a higher number of bits results in more precise input values for the decoder. When using hard input decision this only matters in case the physical value is close to zero and the bit decision is not correct. It is known the soft input values are represented in LLR. The absolute value of the LLR is used to select the surviving path. Therefore in soft input decision decoding the quantization error is more likely to influence the path decision.

It is very natural that a higher number of bits in the demapper and at the input of the decoder increases the error correction ability. This way, additional information is given from the transmission channel. In contrast to code parameters the intput bit length does not affect the convolutional code itself, nor the algorithm of encoding and decoding has to be changed. Furthermore, there also is no effect on the structure of hardware implementation. But a higher number of bits for soft input will increase the amount hardware. Therefore it is desirable to use as few bits as possible at the input, but as much as needed to have a good error correction ability.

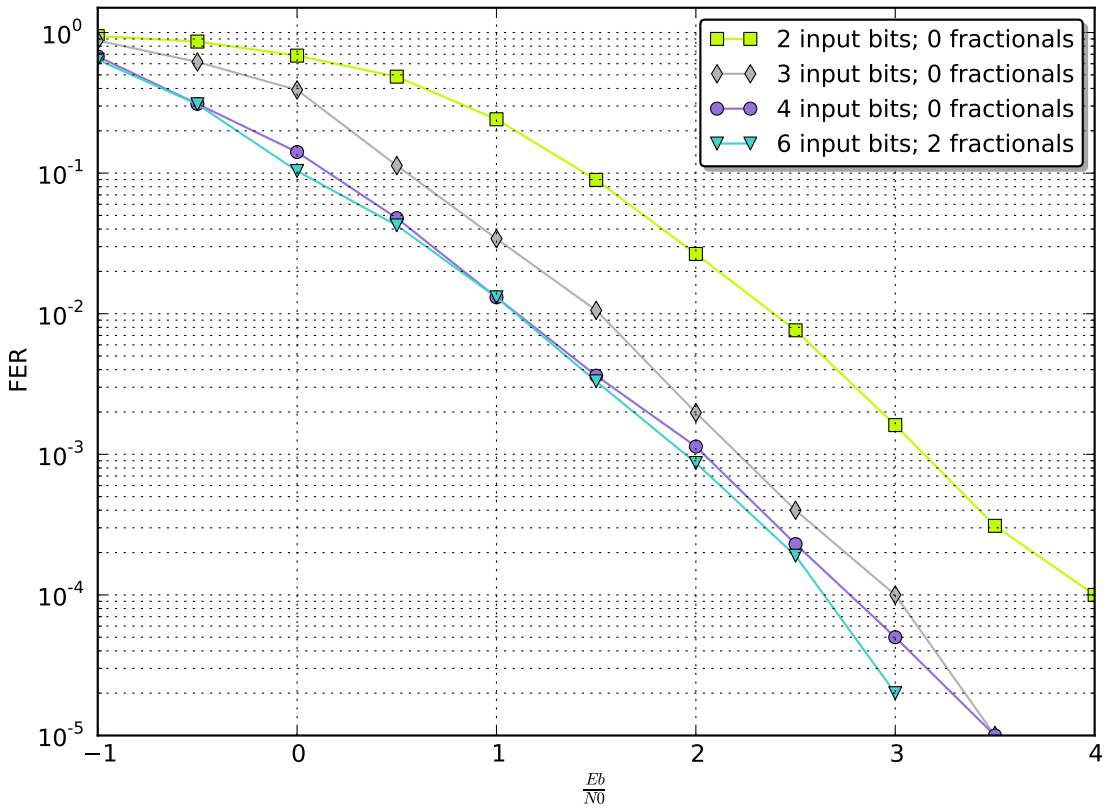

Figure 4.8: Variable input bit length; Rate = 1/3; Constraint length = 9; Frame size = 228

As already claimed the error correction ability depends on the precision of the LLR value at the input. This shall be prospected with a simulation, where different bit width are observed. Since the convolutional code itself does not influence the behavior, standard parameters from CDMA 2000 are selected. Rate $1/3$ is

chosen, which results in polynomial set $[557, 663, 711]$, while the frame size is set to 228 and no windowing is used. Figure 4.8 shows different bit lengths, while fractional resolution of channel symbols is always to the best value for a given input bit length.

At first it can be observed, withean increasing number of bits the error correction ability is increased. While using 2 or 3 bits is worse than using 4 bits input value representation, it is also easy to see the gain in error correction is getting less. Using more bits at the decoder input also goes along with using more fractional data of the channel symbols. This gives more accurate LLR values with a smaller quantization error, but at some point the ability of decoding is at its limit.

On a second view it can be observed, that the gain from using more input bits is getting more when simulating a high SNR range. In high SNR ranges a smaller bit length is not capable to represent the given data and will cap the value at some point. Using higher bit length here helps to distinguish between different input values.

This way the usage of bit length larger than 4 does not result in a much better error correction ability in low SNR rates, but in higher. In most real case scenarios very high SNR ranges are not likely. Therefore using 4 bit representation for LLR values, while using no fractional information from channel symbols is a sufficient choice. Using higher a bit length will not decrease the error correction ability, but the gain is not worth the effort in hardware usage for demapper and decoder.

In conclusion, Viterbi decoding works efficient for small input bit lengths.

4.5 Standard Comparison

The previous sections show the error correction ability depending on different parameters. Depending on the application and needs the standards define different sets of parameters. Therefore it is difficult to compare them directly.

The following simulation present an overview, of the error correction ability in different standards. In order to get comparable results the windowing technique from Section 4.3 is not used here and the soft decision input bit length from Section 4.3 is set to 4 bits. Furthermore the frame length from Section 4.2 is set to 400, which is a valid frame length for most standards. Table 1.2 shows the used codes and Figure 4.9 presents the simulation results. Based on results from previous simulation Chapter 4, there is no surprising result.

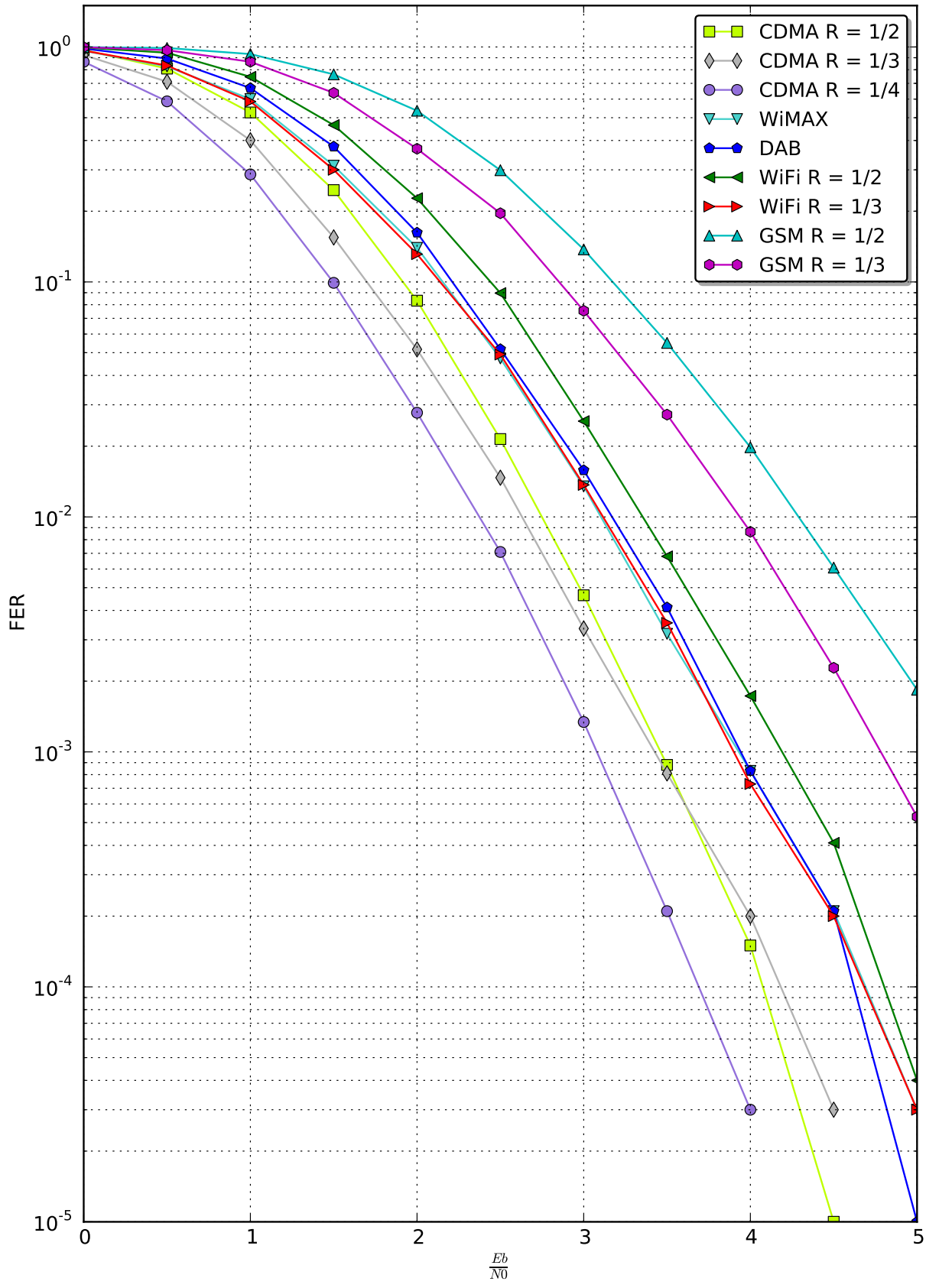

Figure 4.9: Standards overview; Frame length = 400; Input bit length = 4; No windowing

5 Version Information

5.1 Product Version

Version	Date	Comment
1.0.0	12/01/16	Initial version.

5.2 Document Versions

Version	Date	Comment
1.0.0	12/01/16	Initial version.

Bibliography

[1] ARM. AMBA 4 AXI4-Stream Protocol, March 2010.