Features

• Drop-in module for Spartan™-3, Spartan-3E, Spartan-3A/3AN, Virtex™-II, Virtex-II Pro, Virtex-4, and Virtex-5 FPGAs
• Implements the CDMA2000/3GPP2 specification [1]. Core contains the full 3GPP2 interleaver
• Full range of 3GPP2 block sizes supported, (122-12282)
• Core implements the MAX*, MAX, or MAX SCALE algorithms
• Dynamically selectable number of iterations 1-15
• Number representation: two’s complement fractional numbers
• Data input: 2 to 5 integer bits and 1 to 4 fractional bits
• Internal Calculations: 6 to 9 integer bits and 1 to 4 fractional bits
• Sliding window size of 32 or 64
• Works with all 3GPP2 code rates
• Internal or external RAM data storage
• To be used with Xilinx CORE Generator™ system

Applications

This version of the Turbo Convolution Code (TCC) decoder is designed to meet the 3GPP2 mobile communication system specification [1].

LogiCORE Facts

<table>
<thead>
<tr>
<th>Core Specifics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Device Family</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Provided with Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
</tr>
<tr>
<td>Design File Formats</td>
</tr>
<tr>
<td>Verification</td>
</tr>
<tr>
<td>Instantiation Template</td>
</tr>
</tbody>
</table>

Design Tool Requirements

| Xilinx Implementation Tools | ISE 9.1i or later |

Licensing

Pay Core. Requires a full or evaluation license

Support

Provided by Xilinx, Inc.
General Description
The TCC decoder is used in conjunction with a TCC encoder to provide a reliable, extremely effective way to transmit data over noisy data channels. The Turbo Decoder core operates very well under low signal-to-noise conditions and provides a performance close to the theoretical optimal performance as defined by the Shannon limit [2].

When a decoding operation is started, the core accepts the block size and the number of iterations from two input ports. The systematic and parity data is read into the core in parallel on a clock-by-clock basis. The core then starts the decoding process and implements the required number of decode iterations.

Finally, the decoded bit sequence is output. The entire sequence is automatically controlled from a single first data signal and requires no user intervention. In addition, all the interleaving operations required in the 3GPP2 specification are handled automatically within the core.

The core expects two's complement fractional numbers as inputs and also uses this format for the internal calculations. Each fractional input number represents the Log Likelihood Ratio (LLR) divided by 2 for each input bit. This LLR value can be considered to be the confidence level that a particular bit is a one or zero. The user can trade off accuracy against speed and complexity by selecting the numerical precision that is required. The input data can have 2 to 5 integer bits and between 1 and 4 fractional bits. The precision of the internal calculations can also be controlled with 6 to 9 integer bits and between 1 and 4 fractional bits. The number of internal integer bits must be greater than the number of input integer bits by 3 or more and the number of input fractional bits must be less than or equal to the number of internal fractional bits.

Algorithm Type
The full TCC decoder algorithm is extremely computational and, therefore, approximations must be made to make the algorithm usable in practice. The approach taken here is to provide the user with three algorithm choices:

1. **MAX*. A very good algorithmic approximation used when accuracy, rather than algorithm simplicity, is required. BER performance of this approach is the best of all three algorithms although this increases core complexity and resource requirements. In this algorithm, a small lookup table is used to increase the accuracy of some non-linear operations.

2. **MAX.** Produces lower BER performance than the MAX* algorithm, but provides the advantage of being less complex and, therefore, requires fewer resources. In this case, the lookup table is not used, which reduces the algorithm accuracy and subsequently produces a slightly degraded BER performance (approximately 0.5 dB compared to the MAX* algorithm).

3. **MAX SCALE.** Produces BER performance very close to the MAX* (within approximately 0.1 dB to 0.2 dB) but with the complexity of the MAX algorithm. If the small reduction in BER performance is acceptable, this provides the best BER performance/resource requirement trade-off. Reference [3], Improving the MAX Log MAP Turbo Decoder, describes this approach in greater detail.

Sliding Window
A commonly used technique to reduce the resource requirements of the core is the use of a **sliding window** in the calculations. As the sliding window only stores a subset of the entire data set at any one time, the memory requirements are significantly reduced. Two sliding window sizes can be used with the core: 32 or 64.
**Code Rates**

The core operates with all the different code rates of the 3GPP2 specification and always assumes that rate 1/5 data is used as input. For different code rates, the appropriate parity bits in the sequence are replaced by zeros, allowing the core to implement any puncturing scheme.

**Input/Output Pins**

Signal names are shown in Figure 1 and described in Table 1.

![Figure 1: Input and Output Ports for TCC Decoder](image)

**Table 1: Core Signal Pinout**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Direction</th>
<th>Port Width (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD_IN</td>
<td>Input</td>
<td>1</td>
<td><strong>First Data</strong> When asserted (High) on a valid rising clock edge, the decoding process is started. Qualified by ND when present.</td>
</tr>
<tr>
<td>ND</td>
<td>Input</td>
<td>1</td>
<td><strong>New Data</strong> When asserted (High) on a valid rising clock edge, a new input value is read from the DIN port. Ignored when RFD deasserted.</td>
</tr>
<tr>
<td>ACLR</td>
<td>Input</td>
<td>1</td>
<td><strong>Asynchronous Clear</strong> When asserted (High), the decoder asynchronously resets.</td>
</tr>
<tr>
<td>SCLR</td>
<td>Input</td>
<td>1</td>
<td><strong>Synchronous Clear</strong> When asserted (High) on a valid rising clock edge, the decoder is reset.</td>
</tr>
<tr>
<td>CE</td>
<td>Input</td>
<td>1</td>
<td><strong>Clock Enable</strong> When this is deasserted (Low), rising clock edges are ignored and the core is held in its current state. A rising clock edge is only valid when CE is asserted (High).</td>
</tr>
<tr>
<td>CLK</td>
<td>Input</td>
<td>1</td>
<td><strong>Clock</strong> All synchronous operations occur on the rising edge of the clock signal.</td>
</tr>
<tr>
<td>ITERATIONS</td>
<td>Input</td>
<td>4</td>
<td><strong>Iterations</strong> The number of iterations that the core must implement. Read on a valid FD_IN assertion (High).</td>
</tr>
<tr>
<td>BLOCK_SIZE_SEL</td>
<td>Input</td>
<td>4</td>
<td><strong>Block Size Select</strong> The block size of the current decode operation. Read on a valid FD_IN assertion (High).</td>
</tr>
</tbody>
</table>
**Functional Description**

**Clock: CLK**

All operations of the core are synchronized to the rising edge of the CLK signal. If the optional CE pin is present, a rising clock edge is only valid when CE is High; if CE is Low, the core is held in its current state.

**Clock Enable: CE**

CE is an optional pin used to indicate if the next rising clock edge is valid. When CE is High, rising clock edges are valid and allow the decoding process to continue. If CE is Low, the core operations are suspended and the core remains in its current state. All synchronous signals are ignored when CE is Low.

**First Data: FD_IN**

FD_IN is used to start the decode operation. When FD_IN is High on a valid rising clock edge, then the first data is read from the DIN port. Simultaneously, the same clock edge is used to read the values of the BLOCK_SIZE_SEL and the ITERATIONS ports which define the block size and the number of iter-
ations to be implemented for this decode operation. FD_IN should only be held High for a single valid clock cycle. If optional pin ND is present then FD_IN is only valid when ND is High for the same valid clock edge.

**New Data: ND**

The ND signal is optional and is used to indicate that there is new input data to be read from the DIN port. For example, if ND is required and the input block size is 122, then 122+6 (tail bits) active High ND assertions are required to load in the complete block before the decoding operation commences. After all the expected input data has been read into the core, the ND signal is ignored until the next decoding block is started.

**Asynchronous Clear: ACLR**

The ACLR signal is optional and when it is asserted High, the core is reset to its initial state, that is, the core is ready to process a new block. Following the initial configuration of the FPGA, the core is automatically in the reset state, so no further ACLR is required before a decoding operation can take place. The highly pipelined nature of the decoder core means that any ACLR signal actually creates some SCLR signals internal to the core. For this reason, the SCLR signal is recommended for use in this core.

**Synchronous Clear: SCLR**

The SCLR signal is optional and when it is asserted High on a valid rising clock edge, the core is reset to its initial state, that is, the core is ready to process a new block. If the optional pin CE is present then SCLR is ignored if CE is Low. Following the initial configuration of the FPGA, the core is automatically in the reset state so no further SCLR is required before a decoding operation can take place.

**ITERATIONS**

The 4-bit input port represents the number of iterations with valid values from 0001-1111 (binary) or 1-15 (decimal). The ITERATIONS port is read when a valid FD_IN occurs. The value read defines the number of iterations to be implemented for that blocks decode operation.

**Block Size Select: BLOCK_SIZE_SEL**

The 4-bit input port represents the 3GPP2 block sizes as detailed in Table 2. Like the ITERATIONS port, BLOCK_SIZE_SEL port is read when a valid FD_IN occurs. The value read defines the size of block to be processed for this decode operation. All other values on BLOCK_SIZE_SEL that are not covered in Table 2 are invalid. (Note that block size values do not include tail bits.)

<table>
<thead>
<tr>
<th>Block Size</th>
<th>BLOCK_SIZE_SEL (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>0000</td>
</tr>
<tr>
<td>250</td>
<td>0001</td>
</tr>
<tr>
<td>506</td>
<td>0010</td>
</tr>
<tr>
<td>762</td>
<td>0011</td>
</tr>
<tr>
<td>1,018</td>
<td>0100</td>
</tr>
<tr>
<td>1,530</td>
<td>0101</td>
</tr>
<tr>
<td>2,042</td>
<td>0110</td>
</tr>
</tbody>
</table>
Data In: DIN

The DIN port, a single-input bus, accepts the five input channels of the systematic and parity data. For example, if 2 integer and 3 fractional bits are required for each soft input value, a total of 5 bits is required for each of the five channels. The total bit width of the DIN port is therefore $5 \times 5 = 25$ bits. The arrangement of the DIN port for this example is shown in Figure 2. In this case, the 25-bit input port is represented by DIN[24:0], indicating that bit 24 is the MSB and bit 0 the LSB. The two parity values from the non-interleaved data are represented by RSC1_0 and RSC1_1, while the two parity values from the interleaved data are represented by RSC2_0 and RSC2_1. Figure 2 shows how the user must map each of the systematic and parity bits to each of the DIN bits.

The data input to the decoder core is assumed to be encoded using a corresponding Turbo Encoder core, such as the Xilinx 3GPP2 Turbo Encoder v2.0. A basic description of the Turbo Encoder core is provided, see "Turbo Encoder" on page 7.

<table>
<thead>
<tr>
<th>Block Size</th>
<th>BLOCK_SIZE_SEL (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,066</td>
<td>0111</td>
</tr>
<tr>
<td>4,090</td>
<td>1000</td>
</tr>
<tr>
<td>5,114</td>
<td>1001</td>
</tr>
<tr>
<td>6,138</td>
<td>1010</td>
</tr>
<tr>
<td>8,186</td>
<td>1011</td>
</tr>
<tr>
<td>12,282</td>
<td>1100</td>
</tr>
</tbody>
</table>

Read Data In: RD_D_IN

This port is only used where an external memory interface is required. This port is always the same width as the DIN port because it is used to read the input data from external memory. It is constructed in the same way as the DIN port in that it consists of the five data channels, that is, the systematic input and four parity data inputs (Figure 2). Note that systematic and parity data is read in different orders during the decoding process, which means that one external memory block is required for the systematic data and a separate block is required for the parity data. These two memory areas are addressed by the S_ADDR and P_ADDR signals, respectively.

Discontinued IP
**Ready For First Data: RFFD**

When this signal is asserted, it indicates that the core is ready to accept an FD_IN signal to start a new decode operation. Immediately after a valid FD_IN signal is detected, the RFFD signal will go Low and remain Low until it is able to start another block.

**Ready For Data: RFD**

When this signal is asserted, it indicates that the core is ready or in the process of accepting a new block of input data. This signal remains High until a block is initiated via a valid FD_IN signal and the corresponding block of data including tail bits has been input.

**Ready: RDY**

This signal is asserted after completing the number of iterations defined by the ITERATIONS port on the valid FD_IN signal. RDY is asserted High to indicate that the data on the DOUT port is now valid and forms the final result of the decode operation. RDY is always asserted for a number of valid clock cycles equal to the size of the current block being decoded.

**Data Out: DOUT**

This is the hard-coded output from the decoding process. Each block of data is output on a serial basis with RDY indicating that the data is valid.

**Systematic Address: S_ADDR**

This 14-bit address port is only used where an external memory interface is required. This port provides the address to the systematic data memory area. When WE is asserted High, data is written to this memory and when WE is Low, data is read.

**Parity Address: P_ADDR**

This is similar to the S_ADDR port except that it provides the address for the parity memory block. Note that there are four soft input parity channels and only a single soft input systematic channel. The parity memory block, therefore, will always be four times larger than the systematic memory block.

**Write Data Out: WR_D_OUT**

This port is only used where an external memory interface is required. This port will always be the same in width and configuration as the DIN and RD_D_IN ports (Figure 2). Data provided by the user on the DIN port appears on the WR_D_OUT port ready for storage in external memory.

**Write Enable: WE**

This port is only used where an external memory interface is required. When the WE signal is High, there is valid data on the WR_D_OUT port that must be written to memory.

**Turbo Encoder**

The data into the DIN port of the Turbo Decoder core must be generated by a TCC Encoder core that provides the correct data format, such as the 3GPP2 Turbo Encoder v2.0 supplied by Xilinx. A brief description of the data output requirements of the Turbo Encoder core is provided here for the purpose of identifying the input requirements of the decoder core.

*Figure 3* shows the basic structure of the Turbo encoder. It consists of two identical Recursive Systematic Convolution (RSC) encoders: one processes the original input data, and the other processes an
interleaved version of the original input data. As a general rule, the original input is delayed by the latency of the interleaver, so that the first and successive outputs from both RSCs are synchronized and occur on the same clock cycle.

The output from each RSC consists of the original input bit or systematic bit and two parity bits that are created by the circuit shown in Figure 1. Some of the RSC bits are not transmitted depending on the selected encoding rate. For example, in rate 1/5 for every one input bit, five output bits are generated.

Figure 4 shows that there is a control at the RSC input that switches between new data input and a feedback input. When block_size values have been output from the encoder, these control switches are switched over to create tail bits, which are used to force the RSCs to a known state. Each of the two RSCs create three sets of soft values during the tail bit generation. The output of the Turbo Encoder (and the input to the Turbo Decoder) always consist of block_size+6 sets of soft values. See the 3GPP2 specification for more details.

![Figure 3: Turbo Convolution Encoder](image)

![Figure 4: Basic Recursive Systematic Convolution (RSC) Encoder](image)
Data Input Sequence

Figure 5 shows the data input sequence for an example case where \textit{block\_size} = 122. The input data consists of the output of the two RSC encoders. The convention used here is that RSC1\_systematic(0) to RSC1\_systematic (121) represent the 122 values generated during the encoding process. The same convention also applies to the parity channels. Figure 5 shows the transition between data input at the end of the block and the tail bit input period. During the RSC1 tail bit input, the RSC2 parity bits are not used and any value on these ports will be ignored by the core at this time. The RSC1 inputs are also ignored during the RSC2 tail bit period, except for the RSC1\_systematic port which is used to input the RSC2\_systematic values. This reduces the number of input bits required by the core.

The example in Figure 5 assumes a rate of 1/5 where all five of the encoder outputs are used. (Note that the RSC2\_systematic data is never transmitted except during the RSC2 tail-bit period.) Where different puncture rates are used, a number of the decoder input values will not be transmitted and these must be set to zero during the decoder input stage. This is straightforward and is left to the user to implement, if required. Refer to the 3GPP2 specification for more details.
Figure 5: Data and Tail Bit Input Sequence into the Decoder Core
Data Input Format

The input to the DIN port of the decoder is proportional to the Log Likelihood Ratio (LLR) of the received data, that is, LLR(x).

\[
\text{LLR}(x) = \ln \left( \frac{\Pr(x=1)}{\Pr(x=0)} \right)
\]

This is the natural logarithm of the probability that a received symbol is a one or zero. The actual required input of the Turbo Decoder core is LLR(x)/2. Knowledge of the input data format and the noise characteristics is required to calculate LLR(x) accurately. The Turbo Decoder still functions if an estimate of the LLR is made, but best performance is obtained with an accurate calculation of the LLR.

Assume that there is a Binary Phase Shift Keying (BPSK) input signal, x, where a value of 1.0 represents a transmitted logic 1, and -1.0 represents a transmitted logic 0. Also, assuming that the input has been corrupted by Random Gaussian noise with a mean \(\mu\) and a variance \(\sigma^2\) the LLR(x) can be calculated from

\[
\text{LLR}(x) = \ln \left( \frac{\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( \frac{(x - \mu)^2}{2\sigma^2} \right)}{\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( \frac{(x + \mu)^2}{2\sigma^2} \right)} \right)
\]

This produces:

\[
\text{LLR}(x) = \frac{2\mu x}{\sigma^2}
\]

meaning that the required input into the DIN port of the decoder is given by:

\[
\text{DIN} = \frac{\text{LLR}(x)}{2} = \frac{\mu x}{\sigma^2}
\]

For a mean of 1, the input to the DIN port of the decoder is simply the values output from the encoder divided by the variance. If for example, the mean of the input signals is 3 instead of 1, the input values are simply divided by 3 so that they are rescaled to have a mean of 1. This ensures optimal operation of the Turbo Decoder.

Data Input Format in Relation to Eb/No

It is common practice to measure the error correction performance of different algorithms using the standard measurement:

\[
\frac{\text{Eb}}{\text{No}} = \frac{\text{Energy per bit}}{\text{Noise Density}}
\]

It can be shown that for white Gaussian noise, Eb/No can be related to the noise input by

\[
\frac{\text{Eb}}{\text{No}} = 10 \times \log \left( \frac{1}{2 \times \text{rate} \times \sigma^2} \right) \text{dB}
\]

Plotting the noise variance against Eb/No produces the results shown in Figure 6. This figure shows that with the noise variance set to 1, the decoder only operates optimally at Eb/No values of 0 dB,
1.8 dB, and 3 dB for rates 1/2, 1/3, and 1/4, respectively. For example, if the user requires the decoder to operate at different Eb/No values, the user can either calculate the variance value in real time or use Figure 6 to provide an estimate of the variance. For example, if the decoder is expected to operate when Eb/No is around 2 dB, then appropriate variance values are 0.63, 0.95, and 1.25 for rates 1/2, 1/3, and 1/4, respectively.

For the purposes of this data sheet, when Bit Error Rate (BER) figures are quoted as using scaled values, it is assumed that the noise variance is accurately known. The less accurate the noise variance of the input, the greater the degradation in BER performance.

![Figure 6: Plot of Noise Variance against Eb/No for Different Rates](image)

**External Memory Interface**

The raw input data from the DIN port is used during each decoding iteration, so a RAM must be provided to store the data. This data store can be internal or external to the core.

If the data memory is internal to the core no user action is required. The core handles all storage and addressing functions internally. If the memory storage is external to the core, the external memory ports are used to create the memory interface. These ports are defined in a previous section, but additional detail is included in this section.

Assuming that a 25 bit wide DIN port has been selected (as shown in Figure 2), Figure 7 shows the detail of how the external memory ports are connected. The systematic and parity memory blocks are assumed to be generic memory blocks. If these are external to the FPGA, then it is likely that there is a
single bi-directional data port on the memories, rather than the separate ports shown in Figure 7. These simple changes are left for the user to implement, as they will be specific to the design.

Note that the systematic memory is only 5 bits wide in this example, compared to the parity memory which is 20 bits wide. The parity memory is always 4 times as wide as the systematic memory due to the fact that there are four parity input channels and only one systematic. The maximum block size in the 3GPP2 specification is 12288 (including 6 tail bits), giving a total memory requirement for this example as follows:

- Systematic Memory requirement = 5 bits x 12288 = 61,440 bits
- Parity Memory = 4 channels x 5 bits x 12288= 245,760 bits

It is important to note that the systematic memory is read in a linear and an interleaved sequence. This memory block must therefore be capable of true random access at the full system clock rate. Parity memory is always addressed as an increasing count from zero to block size + tail bits.

**Signal Timing**

The Turbo Decoder core is a synchronous core operating on the rising edge of the clock. All input signals are read and all output signals can be changed on the rising edge of the clock. The only exception to this is the asynchronous clear signal, ACLR. When an optional CE signal is used, the core state does not change when CE is Low; all input signals are ignored and the core outputs remain the same. If the optional CE signal is not used, the core operates as though CE is permanently High (enabled).

Figure 8 shows the input timing for the decoder. The data input process is started when FD_IN, CE and ND are all High on a rising clock edge. On receiving a valid FD_IN pulse, the RFFD signal goes Low to indicate that the core is no longer ready to receive a first data pulse. RFFD remains Low until the core is ready to process another block of data.

The first input data, d0, is read from the DIN port on the same clock edge as the valid FD_IN pulse is detected (Figure 8). At the same time, the ITERATIONS and BLOCK_SIZE_SEL inputs are read to determine the size of block to be processed and the number of decode iterations to be implemented.

The core will read the next input data values on successive rising edges of the clock unless CE or ND is Low, in which case the input data is ignored. During the data input process the RFD signal remains High to indicate that the core is ready to accept further input data.
As shown in Figure 9, at the end of the input cycle, after BLOCK_SIZE+6 data values have been input, the RFD signal will go Low to indicate that all input data has been read. Once RFD is Low further ND signal changes are ignored. The RFD signal going Low also indicates that the core is moving from its input to its decoding phase.

After the decoder has performed the required number of iterations, the RDY signal is driven High to indicate that there is valid data on the DOUT port (Figure 10).

When the core approaches the end of the data output phase, it takes RFD and RFFD High to indicate that the core is ready to accept a new block. These signals go High before the last data has been output to maximize throughput.
Performance and Resource Usage

The core has been extensively tested to optimize performance. Information is provided on both performance and resource use to allow the user to make informed design decisions based on the requirements of their application.

BER Performance

The effect of different block sizes, rates and other parameters on the BER performance of the core has been measured and plotted against Eb/No. This allows the designer to determine the optimum trade off between core performance and complexity. Table 3 indicates which parameter is varied in each of the BER Performance Plots shown in Figure 11 through Figure 15.

Table 3: BER Performance Plots

<table>
<thead>
<tr>
<th>Plot BER vs. Eb/No</th>
<th>Parameter Being Varied</th>
</tr>
</thead>
<tbody>
<tr>
<td>See Figure 11</td>
<td>Block size</td>
</tr>
<tr>
<td>See Figure 12</td>
<td>Code rates</td>
</tr>
<tr>
<td>See Figure 13</td>
<td>Algorithms</td>
</tr>
<tr>
<td>See Figure 14</td>
<td>Iterations</td>
</tr>
<tr>
<td>See Figure 15</td>
<td>Window size</td>
</tr>
</tbody>
</table>

The core configuration, code rate, block size used, and number of iterations implemented for each trace in the BER plots are identified in the legend. For example:

1r3,2i3,6m3, scale, w32, bs122, i5

Where:

1r3 = code rate 1/3
2i3 = 2 input integer bits and 3 fractional input bits
6m3 = 6 metric integer bits and 3 fractional metric bits
scale = max scale algorithm (alternatively, star = max star algorithm)
w32 = window size of 32 (alternatively, w64 = window size of 64)
bs122 = block size of 122 excluding tail bits
i5 = 5 iterations

These results have been generated in hardware using a Virtex-4 device. The device was configured with a setup consisting of an encoder, noise channel, and the decoder. Additional logic in the FPGA was used to record both the throughput and the bits in error between the input data to the encoder and the output data from the decoder.

The input data shown in the plots has been scaled as described in the Data Input Format section.
Resource Requirements and Core Performance

The resource requirements and associated core performance have been given for Virtex-4 and Virtex-5 FPGAs in Table 4 and Table 5, respectively. A total of four core configurations have been considered for both devices. Each case is identified below where the bit widths are represented in a similar way to that described on the previous page.

- Case A: 2i3, 6m3, max-scale algorithm, no optional ports
- Case B: 2i3, 6m3, max-scale algorithm, with optional ports CE, ND and SCLR
- Case C: 2i3, 6m3, max-star algorithm, no optional ports
- Case D: 5i4, 9m4, max-scale algorithm, no optional ports

A window size of 32 and internal data memory has been used for all of the above cases. Note that both resource use and the maximum achievable clock rate are irrespective of the block size, code rate used, or number of iterations implemented.

Table 4: Performance and Resource Requirements for Virtex-4 XC4VSX25 (speed grade -10)

<table>
<thead>
<tr>
<th>Case</th>
<th>Max. Clock Rate</th>
<th>LUTs</th>
<th>Flip-Flops</th>
<th>Hardware Multipliers</th>
<th>Block RAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>130 MHz</td>
<td>2,985</td>
<td>1,617</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>B</td>
<td>126 MHz</td>
<td>3,204</td>
<td>1,678</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>C</td>
<td>110 MHz</td>
<td>3,474</td>
<td>1,743</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>D</td>
<td>122 MHz</td>
<td>3,886</td>
<td>2,135</td>
<td>4</td>
<td>43</td>
</tr>
</tbody>
</table>

Notes:
1. Area and maximum clock frequencies are provided as a guide. They may vary with new releases of Xilinx implementation tools, etc.
2. Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification.

Table 5: Performance and Resource Requirements for Virtex-5 XC5VSX35 (speed grade -1)

<table>
<thead>
<tr>
<th>Case</th>
<th>Max. Clock Rate</th>
<th>LUTs</th>
<th>Flip-Flops</th>
<th>Hardware Multipliers</th>
<th>Block RAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>150 MHz</td>
<td>2,496</td>
<td>1,541</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>148 MHz</td>
<td>2,572</td>
<td>1,590</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>122 MHz</td>
<td>3,060</td>
<td>1,658</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>D</td>
<td>144 MHz</td>
<td>3,318</td>
<td>2,030</td>
<td>4</td>
<td>23</td>
</tr>
</tbody>
</table>

Notes:
1. Area and maximum clock frequencies are provided as a guide. They may vary with new releases of Xilinx implementation tools, etc.
2. Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification.

The core latency in clock cycles is given using the following formula:

\[
\text{Core Latency} = (\text{window size}+3)+2*\text{iterations}*(2*\text{window size}+19+\text{block size})
\]

The core latency is defined as the number of clock cycles measured from the first input to the core (that is, when FD_IN is active) to the first decoded data output from the core (when RDY is asserted). RFFD is asserted prior to RDY being deasserted indicating that another block’s decode operation can be initiated before the previous block’s data has been completely output.
The number of clock cycles between successive decode operations is given by the formula:

\[
\text{Clock cycles between blocks} = \text{Core Latency} + \text{block size} - 109 - 3(\text{window size}-32)
\]

Given the number of clock cycles defined in the above equation and the maximum clock frequency of the core, it is possible to calculate the throughput achievable. Table 6 gives some typical values for throughput of the core for a selection of block sizes and numbers of iterations. These assume a window size of 32 and a performance equal to that of a Virtex-5 XC5VSX35 device implementing Case A (150 MHz from Table 5).

**Table 6: Throughput Rates (Mbits/s)**

<table>
<thead>
<tr>
<th>Block Size</th>
<th>3 iterations</th>
<th>5 Iterations</th>
<th>7 Iterations</th>
<th>9 Iterations</th>
<th>11 Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>14.3</td>
<td>8.7</td>
<td>6.3</td>
<td>4.9</td>
<td>4.0</td>
</tr>
<tr>
<td>506</td>
<td>19.1</td>
<td>12.0</td>
<td>8.7</td>
<td>6.9</td>
<td>5.7</td>
</tr>
<tr>
<td>762</td>
<td>19.8</td>
<td>12.5</td>
<td>9.1</td>
<td>7.2</td>
<td>5.9</td>
</tr>
<tr>
<td>1,530</td>
<td>20.6</td>
<td>13.0</td>
<td>9.5</td>
<td>7.5</td>
<td>6.2</td>
</tr>
<tr>
<td>3,066</td>
<td>21.0</td>
<td>13.3</td>
<td>9.8</td>
<td>7.7</td>
<td>6.4</td>
</tr>
<tr>
<td>6,138</td>
<td>21.2</td>
<td>13.5</td>
<td>9.9</td>
<td>7.8</td>
<td>6.4</td>
</tr>
<tr>
<td>12,282</td>
<td>21.3</td>
<td>13.6</td>
<td>9.9</td>
<td>7.8</td>
<td>6.5</td>
</tr>
</tbody>
</table>

**Figure 11: Measured BER Performance of the MAX_SCALE Algorithm**
Figure 12: Measured BER Performance for Different Code Rates
Figure 13: Comparison of MAX* and MAX_SCALE Algorithms
Figure 14: Measured BER Performance for Different Numbers of Iterations
References


**Figure 15:** Measured BER Performance for Window Size of 32 and 64
Ordering Information

This Xilinx LogiCORE™ product is provided under the terms of the SignOnce IP Site License. For additional information about the core and how to obtain a license for the core, please see the TCC Decoder product page. For pricing and availability of Xilinx LogiCORE products and software, please contact your local Xilinx sales representative or visit the Xilinx Silicon Xpresso Cafe.

France Telecom, for itself and certain other parties, claims certain intellectual property rights covering Turbo Codes technology, and has decided to license these rights under a licensing program called the Turbo Codes Licensing Program. Supply of this IP core does not convey a license nor imply any right to use any Turbo Codes patents owned by France Telecom, TDF, or GET. Please contact France Telecom for information about its Turbo Codes Licensing Program at the following address: France Telecom R&D, VAT/TURBOCODES 38, rue du Général Leclerc 92794 Issy Moulaineaux Cedex 9.

Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/11/03</td>
<td>1.0</td>
<td>Initial Xilinx release.</td>
</tr>
<tr>
<td>04/22/04</td>
<td>1.1</td>
<td>Added further performance data.</td>
</tr>
<tr>
<td>04/28/05</td>
<td>2.0</td>
<td>Added support for Spartan-3E.</td>
</tr>
<tr>
<td>02/15/07</td>
<td>2.1</td>
<td>Updated for version 2.1.</td>
</tr>
</tbody>
</table>