LCD01C Features

LDPC Decoder

- CCSDS compatible
- Rate 223/255 (8160,7136)
- Includes ping–pong input and output memories
- Up to 488 MHz internal clock
- Up to 3.5 Gbit/s with 10 decoder iterations
- 6–bit sign–magnitude input data
- Up to 64 iterations
- Scaled min–sum decoding algorithm
- Optional power efficient early stopping
- Parity check output
- 23,453 6–input LUTs, 166 18KB BlockRAMs. 26,800 Altera ALUTs, 166 M9Ks.
- Asynchronous logic free design
- Free simulation software
- Available as EDIF and VHDL core for Xilinx FPGAs under SignOnce IP License. ASIC, Intel/Altera, Lattice and Microsemi/Actel cores available on request.

Introduction

The LCD01C is a fully compatible CCSDS rate 223/255 (8160,7136) LDPC [1] error control decoder. A regular quasic–cyclic LDPC code with 511x511 square circulants with weight 2 in the parity check matrix is used. There are 2x16 circulants, resulting in a check node degree of 32 and a variable node degree of 4.

In each clock cycle, 12 check nodes (12x32 = 384 messages) or 96 variable nodes (96x4 = 384 messages) are fully decoded. Each iteration requires 86 clock cycles to calculate the check or variable messages plus a 7 clock cycle pipeline delay. The scaled min–sum iterative decoding algorithm [2] is used.

Optional early stopping allows the decoder to reduce power consumption with no degradation in performance. The decoder contains two sets of message memories so that check and variable calculations can be performed in parallel. Two input memories are used to buffer the input data.

Figure 1 shows the schematic symbol for the LCD01C decoder. The EDIF core can be used with Xilinx Foundation or Integrated Software Environment (ISE) software to implement the core. The VHDL core can be used with Xilinx ISE or Vivado software. Custom VHDL cores can be used in ASIC designs.

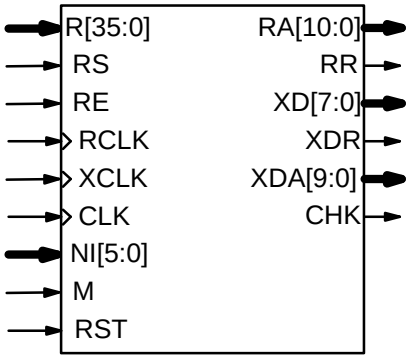

Figure 1: LCD01C schematic symbol.

Table 1 shows the performance achieved with various Xilinx parts. T_{cp} is the minimum clock period over recommended operating conditions. These performance figures may change due to device utilisation and configuration. Note that Zynq devices up to XC7Z020 and from XC7Z030 use programmable logic equivalent to Artix–7 and Kintex–7 devices, respectively.

Table 1: Performance of Xilinx parts.

Xilinx Part	T_{cp} (ns)	Mbit/s
XC7S75–1	6.447	1133
XC7S75–2	5.314	1375
XC7A75T–1	6.341	1152
XC7A75T–2	5.264	1388
XC7A75T–3	4.627	1579
XC7K70T–1	4.607	1586
XC7K70T–2	3.876	1885
XC7K70T–3	3.474	2103
XCKU035–1	4.120	1773
XCKU035–2	3.481	2099
XCKU035–3	2.880	2537
XCKU3P–1	2.620	2789
XCKU3P–2	2.292	3188
XCKU3P–3	2.049	3566

Signal Descriptions

CHK Parity Check
CLK System Clock

M	Early Stopping Mode
NI	Number of Iterations minus one (0–63)
	I = NI+1 wjere I is number of iterations
R	Received Data (eight 6–bit samples)
RA	Received Data Address
RCLK	Received Data Clock
RE	Received Data Enable
RR	Received Data Ready
RS	Received Data Start
RST	Synchronous Reset
XCLK	Decoded Data Clock
XD	Decoded Data
XDA	Decoded Data Address
XDR	Decoded Data Ready

LDPC Decoder Parameters

We model the received sample at time *i* as

$$r_i = A(s_i + n_i) \quad (1)$$

where A is the no–noise amplitude, s_i is the modulated signal with value +1 for coded bit $y_i = 0$ and –1 for $y_i = 1$, n_i is additive white Gaussian noise (AWGN) with normalised variance

$$\sigma^2 = 1/(2RE_b/N_0) \quad (2)$$

and $R = 7136/8160 = 0.8745$ is the code rate.

The value of A directly corresponds to the 6–bit signed magnitude inputs (shown in Table 2). The 6–bit inputs have 63 quantisation regions symmetric about zero. The quantisation regions are labelled from –31 to +31. For example, one could have $A = 15.7$. This value of A lies in quantisation region 16 (which has a range between 15.5 and 16.5). Note that symbols 0 and 32 (+0 and –0) have the same quantisation range of –0.5 to 0.5. For best performance, we recommend $A = 23$.

For input data quantised to less than 6–bits, the magnitude information should be mapped into the least significant bit positions of the input, with the sign bit mapped into the most significant bit. For a symmetric and even number of quantisation levels the least significant bit should always be 1. For example, for 4–level quantisation the input values should be –3, –1, 1 and 3.

Due to quantisation and limiting effects the value of A should be adjusted according to the received signal to noise ratio.

The number of LDPC decoder iterations is determined by *NI*, ranging from 0 to 63. $NI = I–1$ where I is the number of iterations. This is equivalent to 1 to 64 iterations. The decoder initially starts at iteration 0, increasing by one until *NI* is reached or an earlier time if early stopping is enabled.

Table 2: Quantisation for received data.

Decimal	Binary	Range
31	011111	30.5↔∞
30	011110	29.5↔30.5
⋮	⋮	⋮
2	000010	1.5↔2.5
1	000001	0.5↔1.5
0	000000	–0.5↔0.5
32	100000	–0.5↔0.5
33	100001	–1.5↔–0.5
34	100010	–2.5↔–1.5
⋮	⋮	⋮
62	111110	–30.5↔–29.5
63	111111	–∞↔–30.5

The LDPC decoder speed f_d is given by

$$f_d = \frac{F_d K}{(NI + 1.5)(\lceil (N–K)/C \rceil + D)} \quad (3)$$

where F_d is the CLK frequency, $K = 7136$ is the input data length, $N = 8160$ is the number of coded bits, $C = 12$ is the number of parallel check node decoders and $D = 7$ is the decoder pipeline delay. Using these parameters for $NI = 9$ (10 iterations) gives $f_d = 7.308F_d$. Due to synchronisation of the received data to the start of each iteration, the total time taken is *NI*+1.5 iterations on average.

LDPC Decoder Operation

Figure 2 gives a simplified block diagram of the LCD01C decoder. Each of the depth 256 RAMs are configured as simple dual port RAMs with only address locations 0 to 85 and 128 to 213 being used. For a circulant size of 511 and six 6–bit messages being read or written at a time, this implies the first 85 addresses have six messages with the 86th address having only one message. To simplify accessing the messages, the remaining five messages repeat the first five messages.

The basic operation for each RAM involves reading and writing six 6–bit messages in 86 clock cycles. Before the first iteration, received data that was stored at RCLK in the 1st Inputs RAMs are read at CLK into the Check RAMs and 2nd Input RAMs. Check and variable messages are then calculated and stored in succession, reading input data from the the 2nd Input RAMs for the variable message calculations.

Using the two halves of the depth 256 RAMs allows the decoder to work on two sets of received data simultaneously. This allows an effective doubling of the decoder speed.

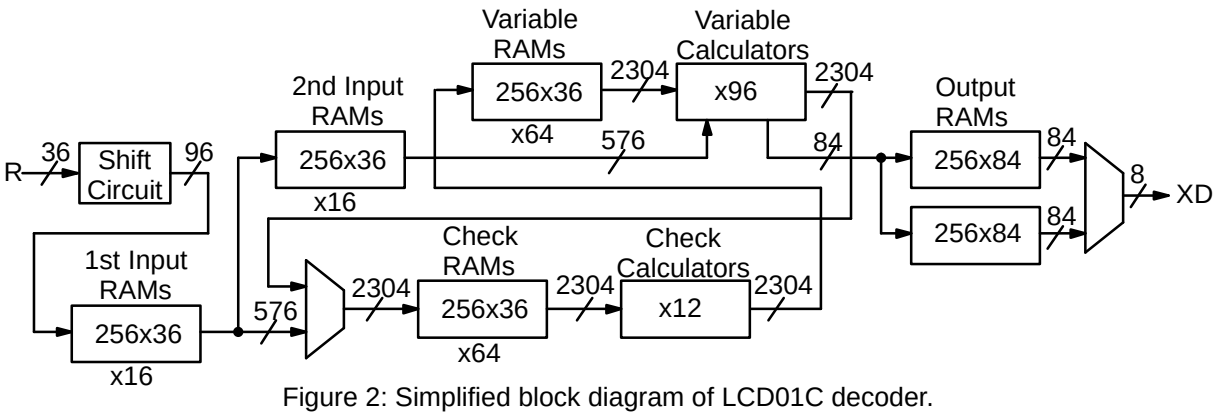

Figure 2: Simplified block diagram of LCD01C decoder.

In order to equalise the pipeline delay of the Check and Variable Calculators to seven clock cycles each, check messages are written and read in sequential order, with the variable messages read and written in deinterleaved order. This is because 511 mod 6 is not zero, which requires complex shifting circuits of the data to perform the read and write of the six messages in the correct order.

Each of the 12 Check Calculators inputs 32 6–bit sign magnitude messages and outputs 32 6–bit two's complement messages. This is performed in five stages using 31 min and next–min circuits. The min circuit finds the minimum of the previous two min values, while the next–min circuit finds the next largest minimum value from the previous two min and next–min values.

A lookup table is used to scale each of the final min and next–min values by 0.8, rounding down to the nearest integer. Rounding down is used to prevent over estimation of the check messages, which results in degraded performance.

The message location of the final min value outputs the final next–min value, with all other 31 message locations using the final min value. The messages are then converted to two's complement, using the XORed sign bits of the 32 input messages.

Each of the 96 Variable Calculators inputs four 6–bit two's complement messages and one received 6–bit sign magnitude value. Two's complement messages are used as this facilitates addition of the messages. The outputs are four 6–bit sign magnitude messages and one hard decision bit if the received data corresponds to a data bit. Since 14 of the circulant array columns correspond to data bits, 14x6 = 84 bits are stored in each clock cycle.

As the basic code is of length 16x511 = 8176, the first 18 bits are set to zero, to produce a length 8158 code. Two dummy 0 bits are then added which increases the code length to 8160. For each iteration, the first three Variable Calculators sets the messages for these bits to 31, ensuring these bits are ignored by the Check Calculators. Note that the multiplexer before the Check RAMs is actually in the Variable Calculators, so that before the first iteration, these bits are correctly set to 31. The last two received values are ignored.

On the last iteration decoded data are stored in the two Output RAMs at CLK. Complex address and multiplexer select generation are then used to read the Output RAMs at XCLK so as to select the decoded 8–bit output in the correct order.

The received data ready RR signal when high indicates when the decoder can accept data. When low, this indicates that new data must not be input to the decoder in the next clock cycle. That is, RS and RE must remain low in the next clock cycle.

If RR is high, received data can be input six 6–bit samples at a time in the next clock cycle. The received data start RS signal is used to start counting. The received data enable RE must be high once and only once for each valid data that is input. Valid data can only be input after RS goes high. A received data address output RA[10:0] is provided for reading received data from an external synchronous read input memory. Data read from the input memory must be held if RE goes low as shown in Figure 3.

When all the input data have been input, RR will go low for 4 RCLK cycles. If the other half of the 1st Input RAMs are available, RR will go high, indicating that the next block may be input. If both halves of the 1st input RAMs are full, then RR will stay low. RR will not go high again until the one of the halves in the 1st Input RAMs becomes available.

If the frequency of RCLK is greater than 1.5 times the frequency of CLK, RR may go high too soon for correct decoder operation. In this case, the waiting period after RR goes low should be one RCLK cycle plus two CLK cycles.

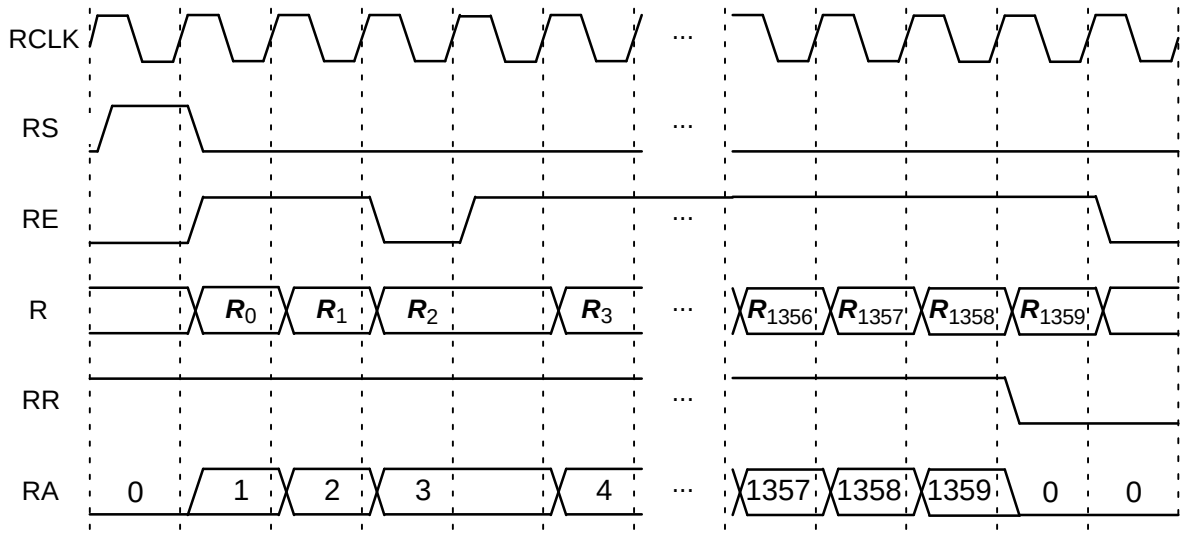

Figure 3: LDPC Decoder Input Timing.

Inputs R[35:0], RS and RE must be synchronous to RCLK. Internal decoding uses CLK.

Figure 3 illustrates the decoder input timing. Each received sample $\boldsymbol{R}_j$, $0 \le j \le 1359$ represents six 6–bit samples R[35:0], $\boldsymbol{R}_j = \{r_{6j+5},...,r_{6j}\}$ and r_i, $0 \le i \le 8159$ corresponds to the received 6–bit sample at time i.

Figure 4 illustrates the decoder output timing. Ignoring the number of clock cycles required to input and output the data, the maximum average number of clock cycles for decoding is $(NI+1)(\lceil (N-K)/C \rceil + D)$. Since two received blocks are decoded in parallel, the decoder delay is $2(NI+1)(\lceil (N-K)/C \rceil + D)$ clock cycles.

The decoded block is output from one of the output RAMs after a block has been decoded. The signal XDR goes high for 892 XCLK cycles while the block is output. The block is output in sequential order with address XDA[9:0]. Note that for the first 8–bit decoded symbol XD[7:0], bit XD[7] corresponds to the first received symbol, that is R[5:0].

There are two decoder operation modes given by M. Mode $M = 0$ decodes a received block with a fixed number of iterations (given by NI). Mode $M = 1$ uses an early stopping algorithm. Early stopping is used to stop the decoder from iterating further once all the parity checks have been satisfied in the block.

For high SNR operation early stopping can lead to significantly reduced power consumption, since most blocks will be decoded with a small number of iterations.

Parity Check Output

The CHK output provides an indication if the parity checks were satisfied during the last iteration. This output is valid while XDR is high. Note that the check is performed before the last variable calculation. It is not performed on the decoded output. If CHK is high this indicates the parity checks were satisfied, indicating that there are probably no errors in the decoded data. If CHK is

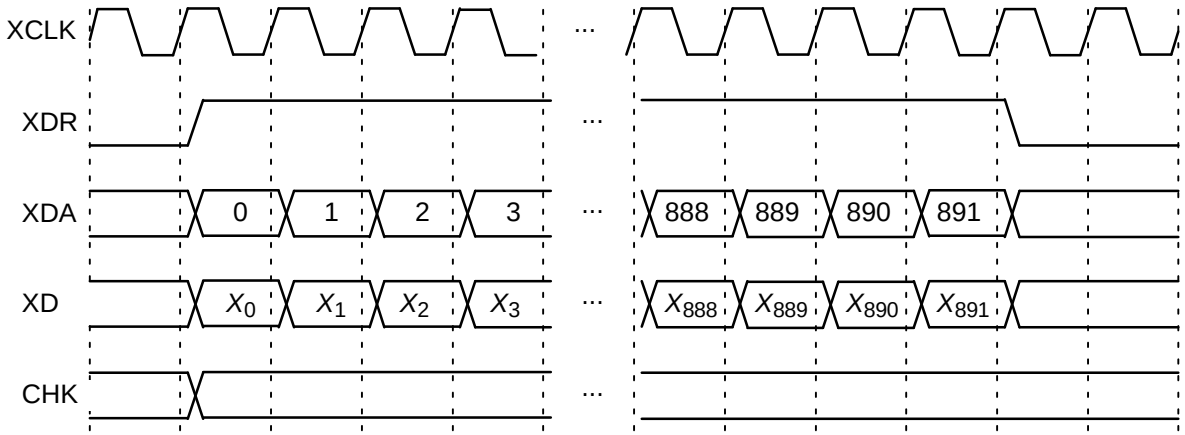

Figure 4: LDPC Decoder Output Timing.

low, this indicates the checks were not satisfied and there are probably errors in the decoded data.

Computer simulations show that the probability of a missed detection, that is the proportion of frames that have errors where CHK = 1 (checks satisfied), is very low. We did not see any events of this type in our simulations. This means that if CHK = 1 it is very likely that there are no errors in the decoder data.

However, the probability of false detection, that is the proportion of frames that have no errors where CHK = 0 (checks not satisfied), can be high. For a low number of iterations or low SNR, the probability is very close to one. That is, nearly all frames that have no errors are falsely detected to have errors. As the number of iterations increases and the SNR increases the probability decreases to zero.

Simulation Software

Free software for simulating the LCD01C LDPC decoder in additive white Gaussian noise (AWGN) or with external data is available by sending an email to info@sworld.com.au with "lcd01csim request" in the subject header. The software uses an exact functional simulation of the LCD01C LDPC decoder, including all quantisation and limiting effects.

After unzipping lcd01csim.zip, there should be lcd01csim.exe and code.txt. The file code.txt contains the parameters for running lcd01csim. These parameters are

`EbNomin`	Minimum E_b/N_0 (in dB)
`EbNomax`	Maximum E_b/N_0 (in dB)
`EbNoinc`	E_b/N_0 increment (in dB)
`optC`	Input scaling parameter (normally 0.75)
`ferrmax`	Number of frame errors to count
`Pfmin`	Minimum frame error rate (FER)
`Pbmin`	Minimum bit error rate (BER)
`state`	State file (0 to 2)
`s1`	Seed 1 (1 to 2147483562)
`s2`	Seed 2 (1 to 2147483398)
`q`	Number of quantisation bits (1 to 6)
`NI`	Number of iterations–1 (0 to 63)
`M`	Stopping mode (0 to 1)
`out_dir`	Output directory
`in_dir`	Input directory
`read_x`	Use external information data (y or n)
`read_r`	Use external received data (y or n)
`out_screen`	Output data to screen (y or n)

The parameter `optC` is used to determine the "optimum" value of A given by

$$A = \frac{\text{optC } Q_{\max}}{\text{mag}(\sigma)} \tag{4}$$

where $Q_{\max}$ = 31 is the maximum input magnitude, σ^2 is the normalised noise variance given by (2) and mag(σ) is the normalising magnitude resulting from an auto–gain control (AGC) circuit.

We recommend using optC = 0.75, which results in A being approximately equal to 23.

We have

$$\text{mag}(\sigma) = \sigma\sqrt{\frac{2}{\pi}}\exp\left(\frac{-1}{2\sigma^2}\right) + 1 - 2Q\left(\frac{1}{\sigma}\right) \tag{5}$$

where $Q(x)$ is the error function given by

$$Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}}\exp\left(\frac{-t^2}{2}\right)dt. \tag{6}$$

Although mag(σ) is a complicated function, for high signal to ratio (SNR), mag(σ) $\simeq$ 1. For low SNR, mag(σ) $\simeq \sigma\sqrt{2/\pi} \simeq 0.798\sigma$. That is, an AGC circuit for high SNR has an amplitude close to the real amplitude of the received signal. At lower SNR, the noise increases the estimated amplitude, since an AGC circuit averages the received signal amplitude.

The simulation will increase E_b/N_0 (in dB) in `EbNoinc` increments from `EbNomin` until `EbNomax` is reached or the frame error rate (FER) is below or equal to `Pfmin` or the bit error rate (BER) is below or equal to `Pbmin`. Each simulation point continues until the number of frame errors is equal to `ferrmax`. If `ferrmax = 0`, then only one frame is simulated.

When the simulation is finished the output is given in file k7136.dat, where K = 7136. The first line gives the E_b/N_0 (`Eb/No`), the number of frames (`num`), the number of bit errors in the frame (`err`), the total number of bit errors (`berr`), the total number of frame errors (`ferr`), the average number of iterations (`na`), and the average BER (`Pb`) and the average FER (`Pf`). Following this, the number of iterations, `na`, `berr`, `ferr`, `Pb`, `Pf`, number of missed detections (`miss`), number of false detections (`fd`), missed detection rate (`Pmiss`) and false detection rate (`Pfd`) are given for each half iteration.

The following file was used to give the BER and FER simulation results shown in Figures 5 and 6, respectively. Auto–stopping was used. When iterating is stopped early, the `nasum` (`num*na`), `berr`, `ferr`, `miss` and `fd` results at stopping are copied for each half iteration to the maximum iteration number. Thus, the I = 10 result is the performance one would measure with auto–stopping and NI = 9.

Figure 7 shows the average number of iterations with E_b/N_0. Figure 8 shows the false detection rate with E_b/N_0.

```
{EbNomin EbNomax EbNoinc optC}
 3.0     5.0     0.1     0.75
{ferrmax Pfmin Pbmin}
 64      1e-99 1e-7
{state s1    s2}
 0     12345 67890
{q  NI  M}
 6  9   1
{out_dir  in_dir  read_x  read_r
out_screen}
 dat      input   n       n       y
```

The `state` input can be used to continue the simulation after the simulation has been stopped, e.g., by the program being closed or your computer crashing. For normal simulations, `state` = 0. While the program is running, the simulation state is alternatively written into State1.dat and State2.dat. Two state files are used in case the program stops while writing data into one file. To continue the simulation after the program is stopped follow these instructions:

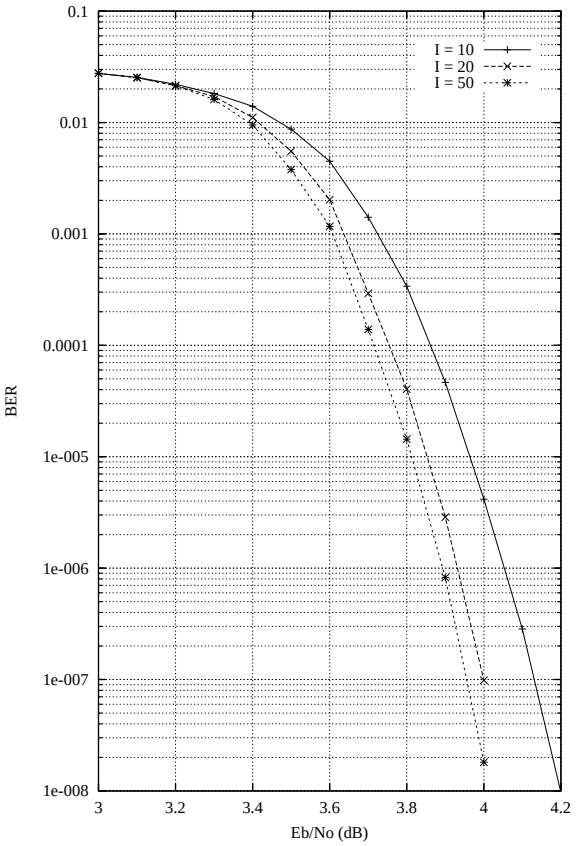

Figure 5: BER performance with auto–stopping and 10, 20 and 50 iterations.

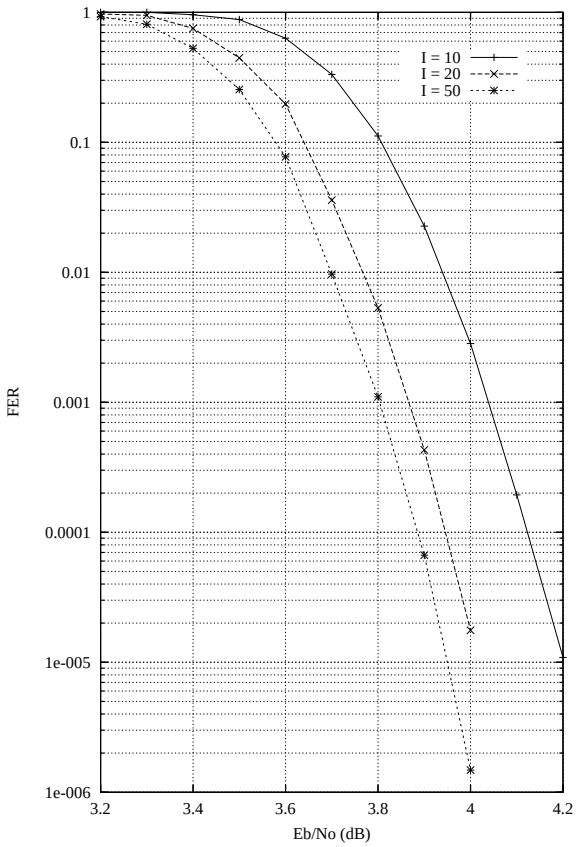

Figure 6: FER performance with auto–stopping and 10, 20 and 50 iterations.

1) Copy the state files State1.dat and State2.dat. This ensures you can restart the program if a mistake is made in configuring code.txt.
2) Examine the state files and choose one that isn't corrupted.
3) Change the state parameter to 1 if State1.dat is used or 2 if State2.dat is used.
4) Restart the simulation. The output will be appended to the existing k7136.dat file.
5) After the simulation has been completed, make sure that `state` is changed back to 0.

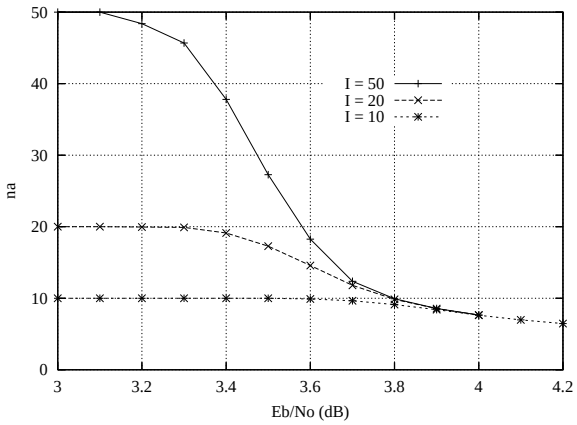

Figure 7: Average number of iterations.

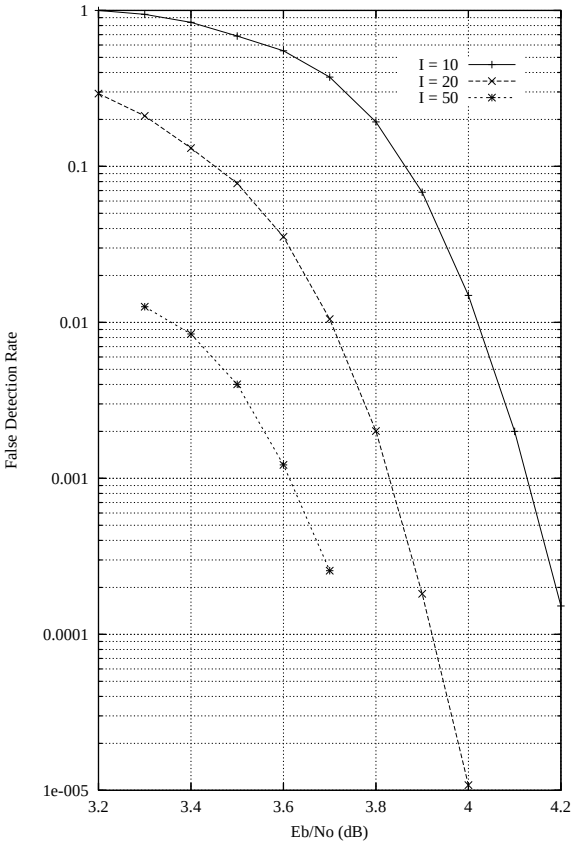

Figure 8: False detection rate with auto–stopping and 10, 20 and 50 iterations.

The software can also be used to encode and decode external data. To encode a block `x_7136.dat` in the directory given by `in_dir`, set `read_x` to `y`, e.g., `x_7136.dat` in directory `input` (each line contains one byte of data in hexadecimal with the left most bit corresponding to the first encoded bit). The encoded stream `y_7136.dat` will be output to the directory given by `out_dir`.

To decode data, place the received block of data in file `r_7136.dat` in directory `in_dir` and set `read_r` to `y`. The decoded data is output to `xd_7136.dat` in directory `out_dir`. `r_7136.dat` has in each line R[i], i = 0 to *N*–1 in decimal form, e.g., the first three lines could be

–25
9
31

The input data is of the form

R[i] = A*(1–2*Y[i]+N[i])

where A is the signal amplitude, Y[i] is the coded bit, and N[i] is white Gaussian noise with zero mean and normalised variance σ^2. For *Q* odd, the magnitude of R[i] should be rounded to the nearest integer and be no greater than $Q_{\max}$.

Ordering Information

SW–LCD01C–SOS (SignOnce Site License)
SW–LCD01C–SOP (SignOnce Project License)
SW–LCD01C–VHD (VHDL ASIC License)

All licenses include EDIF and VHDL cores. The SignOnce and ASIC licenses allows unlimited instantiations. The EDIF core can be used for Virtex–2, Spartan–3 and Virtex–4 with Foundation or ISE software. The VHDL core can be used for Virtex–5, Spartan–6, Virtex–6, 7–Series, UltraScale and UltraScale+ with ISE or Vivado software.

Note that *Small World Communications* only provides software and does not provide the actual devices themselves. Please contact *Small World Communications* for a quote.

References

[1] Consultative Committee for Space Data Systems, “TM synchronization and channel coding,” CCSDS 131.0–B–2, Aug. 2011.

[2] J. Chen and M. P. C. Fossorier, “Near optimum universal belief propagation based decoding of low–density parity check codes,” *IEEE Trans. Commun.*, vol. 50, pp. 406–414, Mar. 2002.

Small World Communications, 6 First Avenue, Payneham South SA 5070, Australia.
info@sworld.com.au ph. +61 8 8332 0319
http://www.sworld.com.au fax +61 8 7117 1416

Revision History

- v0.00 16 Aug. 2012. Preliminary product specification.
- v0.01 26 Oct. 2012. Changed input from R[47:0] to R[35:0]. Increased decoder speed by 50%.
- v0.02 4 Feb. 2013. Updated Altera complexity. Deleted SYNC input. Added XCLK input. Updated decoder operation.
- v1.00 23 Mar. 2013. Added performance of Xilinx parts and BER performance simulations. Added simplified block diagaram and its description.
- v1.01 10 May 2013. Added RA[10:0] output. Changed input buffer RAM to shift circuit, reducing wait delay from 16 to 4 RCLK cycles.
- v1.02 22 Jul. 2013. Corrected input and output file names for simulation software.
- v1.03 15 Oct. 2013. Added CHK output. Corrected RA and RR output for synchronous read input memory. Updated Figure 5, simulation descripton and added frame error rate and false alarm rate simulation figures.
- v1.04 25 Nov. 2014. Added `out_screen` parameter for simulation software.
- v1.05 13 Feb. 2015. Corrected decoder speed for $M = 0$ and 1.
- v1.06 19 Feb. 2015. Corrected decoder speed for $M = 0$.
- v1.07 3 Apr. 2015. Updated core to allow full speed decoding with $M = 1$. Updated Kintex–7 decoder speed.
- v1.08 14 Aug. 2023. Deleted Virtex–4, Virtex–5 and Virtex–6 performance. Updated Artix–7 and Kintex–7 performance and complexity. Added Spartan–7, UltraScale and UltraScale+ performance.