LCD03C Features

LDPC Decoder

- CCSDS TC and TM compatible
- Rate 1/2
- Data lengths of TC 64 and 256 or optional TM 1024 bits
- Includes ping–pong input and output memories
- Up to 624 MHz internal clock
- Up to 2.7 Mbit/s with 30 decoder iterations
- 6–bit two's complement input data
- Up to 256 iterations
- Scaled min–sum decoding algorithm
- Optional power efficient early stopping
- Parity check output
- From 305 to 385 LUTs and 4 to 8 18KB Block-RAMs for Xilinx FPGAs
- Available as EDIF and VHDL core for Xilinx FPGAs under SignOnce IP License. Custom ASIC, Intel/Altera, Lattice and Microchip/Microsemi/Actel FPGA cores available on request.
- Free simulation software

Introduction

The LCD03C is a fully compatible CCSDS rate 1/2 TC (128,64), (512,256) [1] with optional TM (2048,1024) [2] LDPC error control decoder. Irregular quasi–cyclic LDPC codes are used. The information data length is given by *K*.

The TC codes have submatrix sizes of 16 and 64, for *K* = 64 and 256, respectively. The circulant size is the same as the submatrix size. The parity check matrix has four rows and eight columns of circulants for each code. For both codes the check degree is 8 with a variable degree of 3 or 5 with a frequency of 1/2 each. In each row of the parity check matrix there are six weight 1 circulants and one weight 2 circulant.

The TM AR4JA [3] code has a submatrix size of 512 and a circulant size of 128. The parity check matrix has three rows and five columns of submatrices for each code. The check degree is 3 or 6 with a frequency of 1/3 and 2/3, respectively. The variable degree is 1, 2, 3 or 6 with a frequency of 1/5, 1/5, 2/5 and 1/5, respectively. Although the submatrices have weights of 1, 2 or 3, the resulting circulants all have a weight of 1.

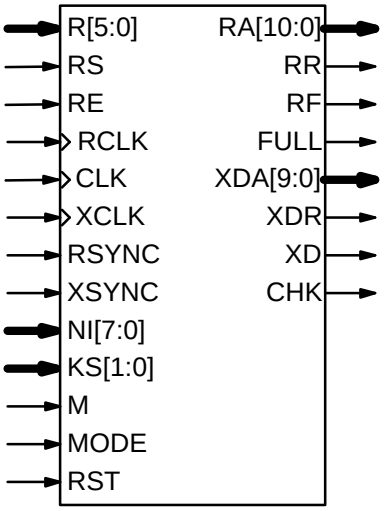

Figure 1: LCD03C schematic symbol.

A fully serial low complexity modified Gauss–Seidel [4] iterative message passing algorithm is used in the decoder. Each iteration requires qMd_c+3 clock cycles to calculate the updated variable messages (VM), where *M* is the submatrix size, d_c is the average check degree value (8 for TC and 5 for TM) and *q* is the number of submatrix rows (4 for TC and 3 for TM). For calculation of the check messages (CM), the scaled min–sum iterative decoding algorithm [5] is used. An input memory is used to buffer the input data.

The LDPC decoder can achieve up to 2.7 Mbit/s with 30 iterations using a 624 MHz internal clock. Optional early stopping allows the decoder to reduce power consumption with no degradation in performance.

Figure 1 shows the schematic symbol for the LCD03C decoder. The VHDL core can be used with Xilinx Integrated Software Environment (ISE) or Vivado software to implement the core in Xilinx FPGA's.

Table 1 shows the performance achieved with various Xilinx parts with 30 decoder iterations. 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: Xilinx Performance (30 iterations).

Xilinx Part	T_{cp} (ns)	f_d (Mbit/s) for K =	
		64 or 256	1024
XC7S6C–1	5.484	0.75	0.81
XC7S6C–2	4.458	0.93	0.99
XC7A12T–1	5.501	0.75	0.80
XC7A12T–2	4.462	0.93	0.99
XC7A12T–3	3.946	1.05	1.12
XC7K70T–1	3.896	1.06	1.14
XC7K70T–2	3.243	1.28	1.37
XC7K70T–3	2.911	1.43	1.52
XCKU035–1	3.090	1.34	1.43
XCKU035–2	2.656	1.56	1.67
XCKU035–3	2.213	1.88	2.00
XCKU3P–1	2.037	2.04	2.18
XCKU3P–2	1.785	2.33	2.48
XCKU3P–2	1.602	2.60	2.77

Table 2 gives the 6–input LUT and 18KB Block-RAM complexity depending on MODE and XSYNC. Note that MODE, XSYNC and RSYNC are soft inputs used to select the decoder configuration. They should not be connected to logic or input pins.

Table 2: Decoder complexity.

MODE	XSYNC	LUTs	BlockRAM
0	0	305	5
0	1	332	4
1	–	385	8

Signal Descriptions

CHK Parity Check
CLK System Clock
FULL Decoder Full (new data not accepted)
KS Data Length Select
0 = 64
1 = 256
2 = 1024
M Early Stopping Mode
0 = No early stopping
1 = Early stopping enabled
MODE Maximum Data Length Select
0 = 64 or 256
1 = 64, 256 or 1024
NI Number of Iterations minus one (0–255)
I = NI+1 wjere I is number of iterations
R Received Data (6–bit)
RA Received Data Address
RCLK Received Data Clock
RE Received Data Enable
RF Received Data Finish
RR Received Data Ready
RS Received Data Start
RST Synchronous Reset
RSYNC RCLK and CLK equal
XCLK Decoded Data Clock
XD Decoded Data
XDA Decoded Data Address
XDR Decoded Data Ready
XSYNC XCLK and CLK equal

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 $R = 0.5$ is the code rate.

The value of A directly corresponds to the 6–bit two's complent input (shown in Table 3) which have 64 quantisation regions. The quantisation regions are labelled from –32 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). For best performance, we recommend $A = 16.5$, 16.5 and 12.5 for KS = 0, 1 and 2, respectively..

Table 3: 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
–1	111111	–1.5↔–0.5
–2	111110	–2.5↔–1.5
⋮	⋮	⋮
–31	100001	–31.5↔–30.5
–32	100000	–∞↔–31.5

Decoder Speed

The number of LDPC decoder iterations is determined by NI, ranging from 0 to 255. $NI = I–1$ where I is the number of iterations. This is equivalent to 1 to 256 iterations. The decoder initially starts at iteration 0, increasing by one until NI is re-

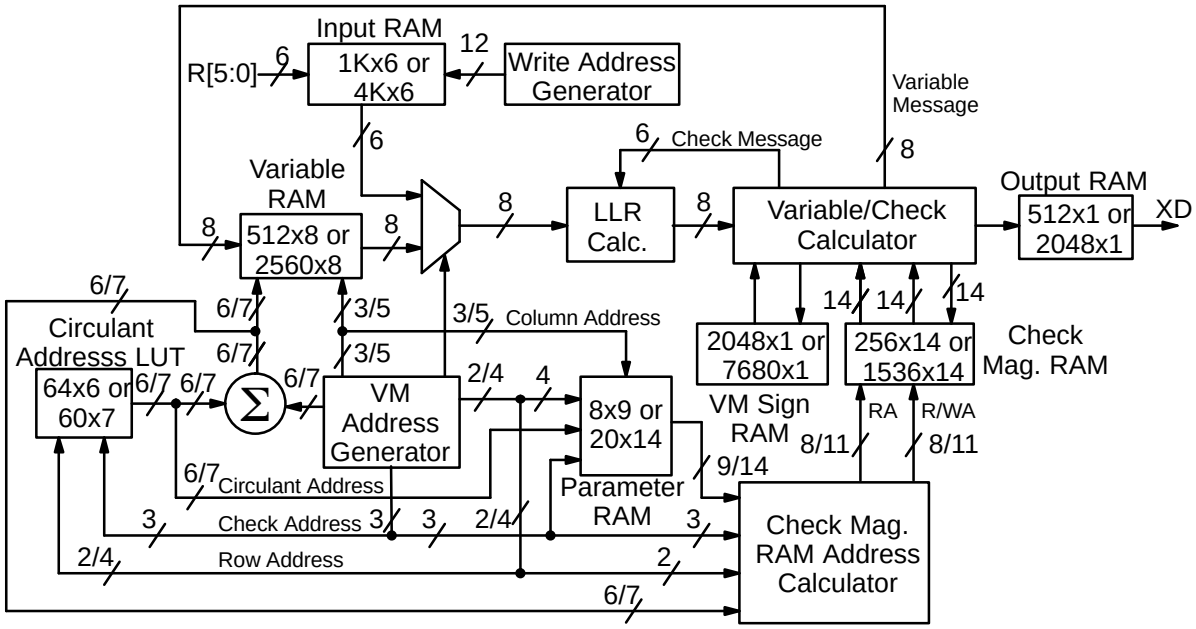

Figure 2: Simplified block diagram of LCD03C decoder.

ached or an earlier time if early stopping is enabled.

The LDPC decoder speed f_d is given by

$$f_d = \frac{F_d K}{IqMd_c + 3} \tag{3}$$

where F_d is the CLK frequency. Using these parameters for NI = 29 (I = 30 iterations) gives f_d = F_d/240.047, F_d/240.012 and F_d/225.003 for K = 64, 256 and 1024, respectively.

Decoder Delay

The decoder delay can be separated into three parts. This is the input memory delay T_i, LDPC decoder delay T_l and the output memory delay T_o. Each delay is equal to

$$T_i = (N + 1)T_r \tag{4}$$

$$T_l = (IqMd_c + 4 - S_r)T_c \tag{5}$$

$$T_o = (3 - 2S_x)T_x \tag{6}$$

where T_r is the RCLK period, S_r = RSYNC, T_c is the CLK period, T_x is the XCLK period and S_x = XSYNC.

The above delays assume that RE is high in the minimum time, no early stopping is used (M = 0) and that the decoder parameters do not change between decoded blocks. When decoding multiple blocks with varying code parameters, the delay is more difficult to predict.

When RSYNC is low the actual LDPC decoder delay will vary from $T_l - T_c$ to T_l. Similarly, when XSYNC is low the actual output delay will vary from $T_o - T_x$ to T_o.

LDPC Decoder Operation

Figure 2 gives a simplified block diagram of the LCD03C decoder. Each memory is implemented using a number of smaller memories that cover the various code options.

The received input data are written 6–bits at a time into one half of the Input RAM. The other half of the memory has 6–bit input data read into the decoder during the first iteration in each clock cycle. The Circulant Address LUT is used to address the Input and Variable RAM, reading the VM for each parity check equation.

For each iteration, for each circulant row i, $0 \le i \le q-1$, the VMs are read for $Md_c(i)$ clock cycles and after a delay of d = 3 clock cycles, the new VMs are written for $Md_c(i)$ clock cycles. At the same time as the new VMs are written, the VMs for the next circulants are read. The order in which the VMs are read and written is carefully arranged so that the VMs for each circulant are read after they are written.

To reduce complexity, there is only one parallel check/variable circuit. Each circuit serially inputs the LLRs for $d_c(i)$ clock cycles. During this time, the previously stored VM sign bits are read for each of the LLR inputs.

The minimum magnitude, next minimum magnitude, VM parity and the address of the minimum magnitude value is read once at the beginning of the calculation. If the address matches the $d_c(i)$ clock address, then the next minimum magnitude is output, otherwise the minimum magnitude is output.

The VM parity, VM sign bit and selected magnitude are combined to form a two's complement CM which is subtracted from the LLR to form an 8–bit VM. This value is stored in the Variable RAM.

The $d_c(i)$ signs of the LLRs are serially exclusive–ORed (added modulo–2) to form the LLR parity check. The magnitude of the VMs (limited to 5–bits each) are used to serially find the minimum magnitude and next minimum magnitude using two comparator circuits. The $d_c(i)$ signs of the VMs are serially exclusive–ORed together to calculate the VM parity.

The VM magnitudes are scaled by 53/64, 49/64 and 51/64 for KS = 0, 1 and 2, respectively. The magnitudes are then rounded down to the nearest integer. This avoids over optimistic values which reduces performance. For KS = 2, check message hard limiting is used to reduce the error floor [6]. If the VM magnitude is greater than or equal to 31, the preliminary check message magnitude (PCMM) is hard limited to 31, otherwise the scaled PCMM is used.

Small lookup tables are used to perform the scaling. The two scaled magnitudes, VM parity and the address of the minimum magnitude are stored in the Check Magnitude RAM. This requires 14 bits. The VM sign bits are stored in the VM Sign RAM.

After the VMs have been read, the previously stored compressed CM values needs to be added to form the LLR. The VM parity is XORed with the sign bit of the VM to produce the sign bit of the CM. This is combined with the magnitudes to produce a two's complement CM. This CM is then added to the VM to produce the new LLR. To avoid over or under flow, positive CMs are added only if the VM is less than 192 and negative CMs are added only if the VM is greater than or equal to –192.

For the Check Magnitude RAM, there are three sets of read and write operations. There is one clock cycle to read the previous iteration CMs and one clock cycle to write the new CMs. This has to be done at the same time as reading d_c CMs for calculation of the new LLRs. A dual port RAM is used to perform this operation.

On the last iteration decoded data are stored in one half of the Output RAM at CLK. Address generation is then used to read the Output RAM at XCLK so as to select the decoded 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 FULL is low, the received data start RS signal is used to start the decoder. The received data enable input RE must also go high when RS goes high to read the first received data. RE can only go high once for each received input symbol. This means RE can only be high for $N = 2K$ clock cycles.

The received data ready output RR will go high to indicated that RE can now go high so as to read the next input. RR will stay high until one clock cycle before the last data is input. RS and RR can be ORed together to form RE if the input data is stored in an external memory.

Valid data must be input one clock cycle after RE 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

The received data finish output RF will go high at the end of each received block. This occurs when RA = N–1. If RE goes low at RA = N–1, RF will remain high. RF will go low only after RE goes high.

If the other half of the Input RAM is available, FULL will remain low, indicating that the next block may be input. If both halves of the RAM are full, then FULL will go high. FULL will not go low again until one of the halves of the RAM becomes available. If FULL is low, RS and RE can go high.

Inputs R[5:0], RS, RE, NI[7:0], KS[1:0] and M must be synchronous to RCLK. Outputs RA[10:0], RR, RF and FULL are synchronous to RCLK. Internal decoding uses CLK. If RCLK and CLK are equal to each other in both clock period and phase, then RSYNC can equal 1. This reduces the decoder input time by one clock cycle. If RCLK and CLK are not equal, then RSYNC must equal 0. RSYNC should not be connected to logic or input pins.

The input data can be input in any code order. That is, it is not necessary to wait for the decoder to output the last block of one code before changing to another code. If changing the code, the decoder parameters NI[7:0], KS[1:0] and M must stay constant from the time RS goes high to until after RF goes high.

Figure 3 illustrates the decoder input timing. Each received sample $\boldsymbol{R}_i$, $0 \le i \le N$–1 represents a 6–bit sample at time i. RS is shown going high again for the case where FULL = 0.

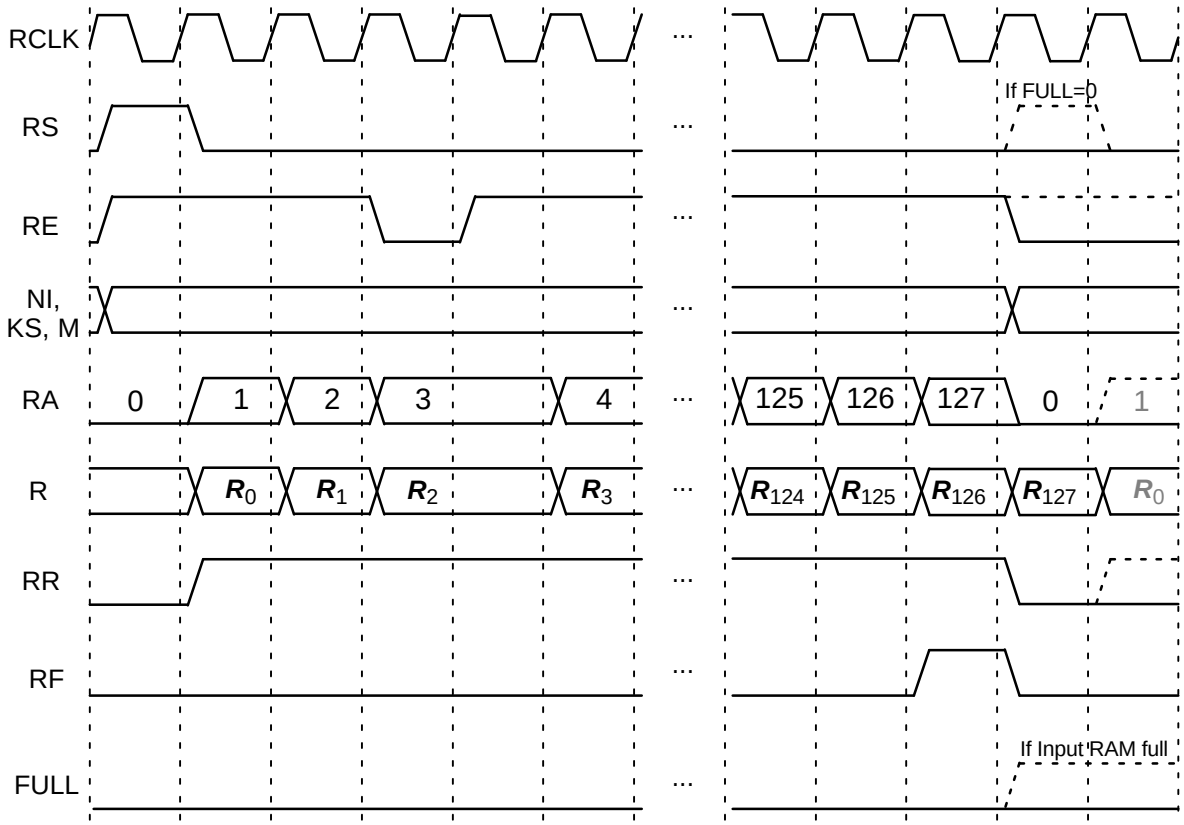

Figure 3: LDPC decoder input timing (KS = 0).

Figure 4 illustrates the decoder output timing. The decoded block is output from one half of the Output RAM after a block has been decoded. The signal XDR goes high for *K*–1 XCLK cycles while the block is output. The block is output in sequential order with address XDA[9:0].

Outputs XD, XDR, XDA[9:0] and CHK are synchronous to XCLK. If XCLK and CLK are equal to each other, i.e., they use the same clock signal, then XSYNC can equal 1. This reduces the decoder output time by two clock cycles. If XCLK and CLK are not equal, then XSYNC must equal 0. XSYNC should not be connected to logic or input pins.

The early stopping algorithm uses the LLR parity checks to determine when to stop. If all the parity checks are satisfied, the decoder will continue for one more iteration and then stop decoding. Early stopping is selected with M = 1. If M = 0, all *I* iterations are performed. 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.

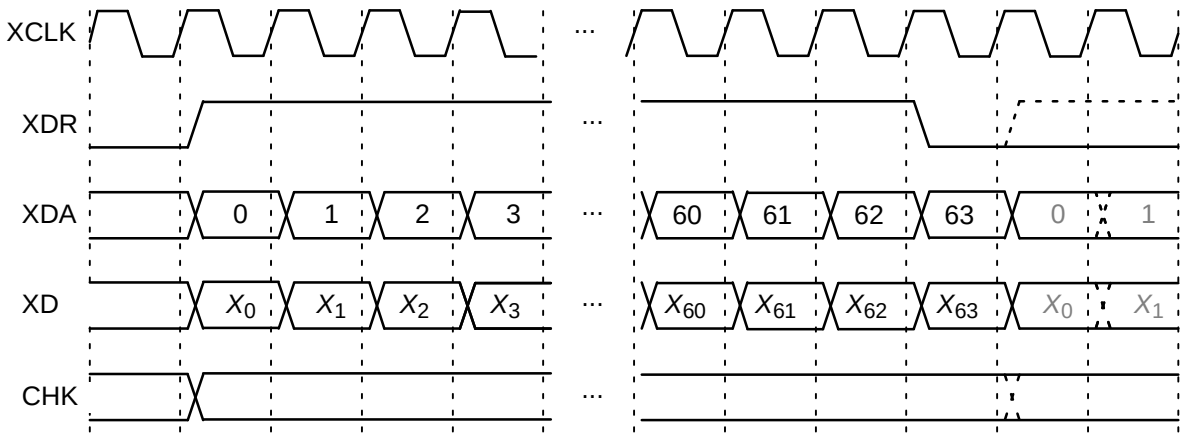

Figure 4: LDPC decoder output timing (KS = 0).

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 LLR 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 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 LCD03C LDPC decoder in additive white Gaussian noise (AWGN) or with external data is available by sending an email to info@sworld.com.au with "lcd03csim request" in the subject header. The software uses an exact functional simulation of the LCD03C LDPC decoder, including all quantisation and limiting effects.

After unzipping lcd03csim.zip, there should be lcd03csim.exe and code.txt. The file code.txt contains the parameters for running lcd03csim. 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)
`A`	Binary modulation amplitude (1–31)
`ferrmax`	Number of frame errors to count
`Pfmin`	Minimum frame error rate (FER)
`Pbmin`	Minimum bit error rate (BER)
`q`	Number of quantisation bits (1 to 6)
`NI`	Number of iterations–1 (0 to 255)
`M`	Stopping mode (0 to 1)
`KS`	Data Length Select (0–2)
`MS`	Message scale (0–64)
`state`	State file (0 to 2)
`s1`	Seed 1 (1 to 2147483562)
`s2`	Seed 2 (1 to 2147483398)
`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)

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` and 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.

If `MS` = `0`, the internal message scale factors are used. For `MS` > `0`, the scale factor `MS/64` is used.

When the simulation is finished the output is given in file k(*K*).dat, for example k64.dat where *K* = 64. 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 simulation results for *K* = 64 with 20 iterations. Figure 5 gives the BER and FER of all three codes with 20 iterations. 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.

```
{EbNomin EbNomax EbNoinc A}
 1.0     5.0     0.5     16.5
{ferrmax Pfmin Pbmin}
 512     1e-0  1e-5
{q NI M KS MS}
 6 19 1 0  0
{state s1    s2}
 0     12345 67890
{out_dir  in_dir  read_x  read_r}
 dat      input   n       n
```

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 com-

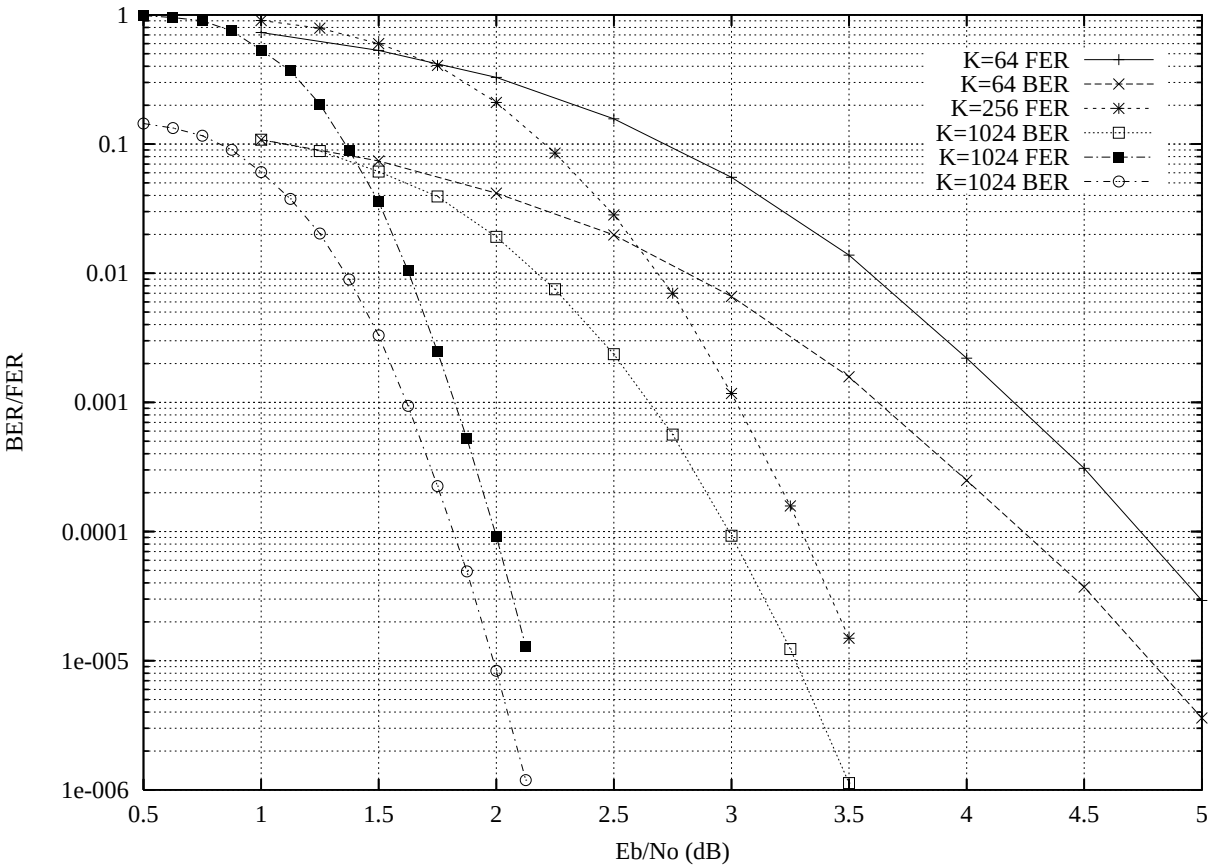

Figure 5: BER and FER performance with auto–stopping and 30 iterations.

puter 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:
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 k(*K*).dat file.
5) After the simulation has been completed, make sure that `state` is changed back to 0.

The software can also be used to encode and decode external data. To encode a block `x_`(*K*)`.dat` in the directory given by `in_dir`, set `read_x` to `y`, e.g., `x_`(*K*)`.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_`(*K*)`.dat` will be output to the directory given by `out_dir`.

To decode data, place the received block of data in file `r_`(*K*)`.dat` in directory `in_dir` and set `read_r` to `y`. The decoded data is output to `xd_`(*K*)`.dat` in directory `out_dir`. `r_`(*K*)`.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–LCD03C–SOS (SignOnce Site License)
SW–LCD03C–SOP (SignOnce Project License)

SW–LCD03C–VHD (VHDL ASIC License)

All licenses include Xilinx 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, “Recommendation for space data system standards: TC synchronization and channel coding,” CCSDS 231.0–B–3, Blue Book, Sep. 2017.

[2] Consultative Committee for Space Data Systems, “Recommendation for space data system standards: TM Synchronization and channel coding,” CCSDS 131.0–B–3, Blue Book, Sep. 2017.

[3] D. Divsalar, S. Dolinar and C. Jones. “Construction of protograph LDPC codes with linear minimum distance,” *IEEE Int. Symp. on Inform. Theory*, Seattle, USA, pp. 664–668, July 2006.

[4] E. Yeo, P. Pakzad, B. Nikolic and V. Anantharam, “High throughput low–density parity–check decoder architectures,” *Global Telecommun. Conf.*, San Antonio, USA, vol. 5, pp. 3019–3024, Nov. 2001.

[5] 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.

[6] J. Hamkins, “Performance of low–density parity–check coded modulation,” *The Interplanetary Network Progress Report*, vol. 42–184, pp. 1–36, 15 Feb. 2011.

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 26 Nov. 2021. Preliminary product specification.
- v0.01 14 Feb. 2022. Changed input to two’s complement. Performanced and implementation results added for KS = 0 or 1.
- v1.00 20 Apr. 2022. Added KS = 2. Updated complexity, performance and Figures 2 and 5. First release.
- v1.01 1 Nov. 2022. Minor updates.
- v1.02 1 Mar. 2024. Added Zynq devices note.