PCD03L8 Features

Turbo Decoder

- 8 state 3GPP™ LTE compatible
- Rate 1/3
- 40 to 6144 bit interleaver
- Up to 280 MHz internal clock
- Up to 204 Mbit/s with 5 decoder iterations
- 6–bit signed magnitude input data
- 1, 2, 4 or 8 parallel MAP decoders
- Optional log–MAP or max–log–MAP constituent decoder algorithms
- Up to 32 iterations in 1/2 iteration steps.
- Optional power efficient early stopping
- Optional extrinsic information scaling and limiting
- Estimated channel error output
- Free simulation software
- Available as VHDL core for Xilinx FPGAs under SignOnce IP License. ASIC, Altera, Lattice and Microsemi cores available on request.

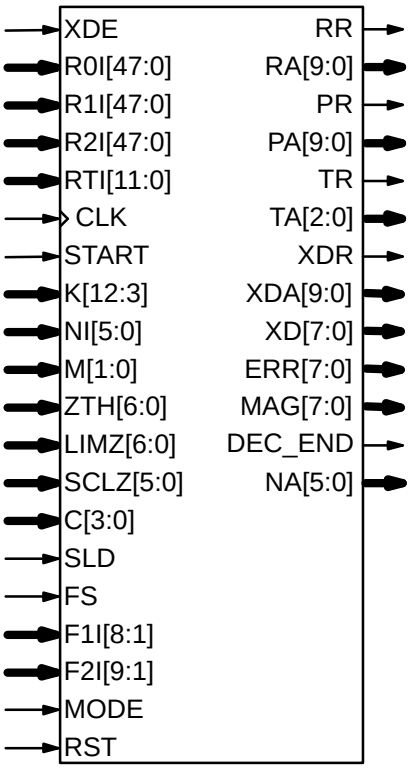

Figure 1: PCD03L8 schematic symbol.

Introduction

The PCD03L8 is a fully compatible 3GPP™ LTE [1] error control decoder. 3GPP™ LTE uses a simple quadratic permutation interleaver.

For LTE, there are 188 interleaver sizes ranging from 40 to 6144 bits. Two parameters f_1 and f_2 are used by the interleaver. All interleaver sizes from 40 to 504 bits that are a multiple of 8, 512 to 1008 bits that are a multiple of 16, 1024 to 2016 bits that are multiple of 32, and 2048 to 6144 bits that are a multiple of 64 are specified. However, the decoder can decode any data length that is a multiple of 8 bits.

For LTE only a code rate of 1/3 is specified. Each tail of the two constituent encoders are terminated using all the data and parity bits (for a total of 12 bits for rate 1/3).

N_p = 1, 2, 4 or 8 MAP03L eight state MAP decoder cores are used with the PCD03L1, PCD03L2, PCD03L4 or PCD03L8 cores, respectively, to iteratively decode the LTE turbo code. The Log–MAP algorithm for maximum performance or the max–log–MAP algorithm for minimum complexity and highest speed can be selected. The extrinsic information can be optionally scaled and limited with each half iteration, improving performance with max–log–MAP decoding.

The reverse sliding window algorithm is used with sliding window lengths of L = 32 or 64 for $K/N_p > L$ and $L = K/N_p$ otherwise. Six–bit quantisation is used for near optimum performance.

The turbo decoder can achieve up to 204 Mbit/s with 5 iterations and max–log–MAP decoding using a 280 MHz internal clock (K = 6144, N_p = 8). Log–MAP decoding decreases speed by about 30%. Optional early stopping allows the decoder to greatly reduce power consumption with little degradation in performance.

Figures 1 to 4 show the schematic symbols for the PCD03L8, PCD03L4, PCD03L2 and PCD03L1 turbo decoders. The VHDL cores can be used with Xilinx Integrated Software Environment (ISE) or Vivado software to implement the core in Xilinx FPGA's.

Table 1 shows the resources used for Kintex–7 devices. Resources for Virtex–5, Virtex–6, Spartan–6 and other 7–Series devices are similar to

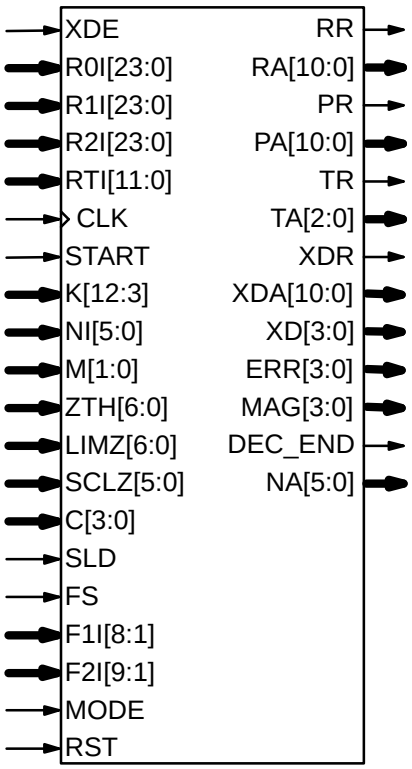

Figure 2: PCD03L4 schematic symbol.

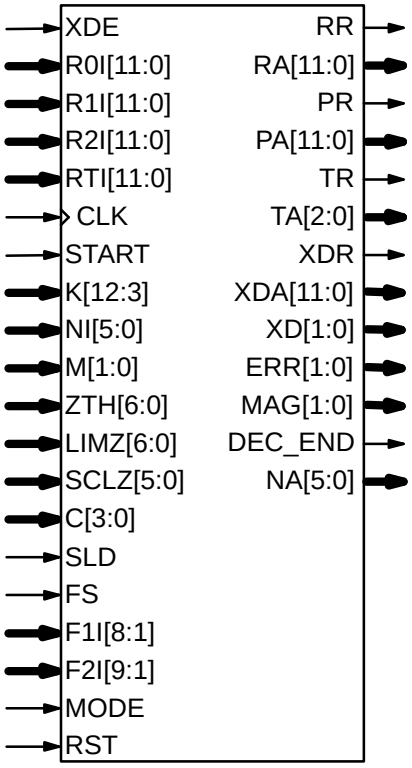

Figure 3: PCD03L2 schematic symbol.

that for Kintex–7. The MODE input can be used to select various decoder implementations. The input/output memory is not included. Only one global clock is used. No other resources are used.

Table 1: Resources used.

MAPs (*Np*)	LUTs Max–log–MAP	LUTs Log–MAP	18KB BRAMs
1	2,160	3,024	3
2	3,888	5,585	3
4	7,170	10,692	3
8	13,817	20,898	4

Table 2 shows the PCD03L8 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.

Table 2: Performance of Xilinx parts.

Xilinx Part	T_{cp} (ns)	*f* (MHz)	f_d (Mbit/s)
XC5VLX50–1	6.843	146.1	106.6
XC5VLX50–2	5.885	169.9	123.9
XC5VLX50–3	5.249	190.5	139.0
XC6VLX75T–1	5.859	170.6	124.5
XC6VLX75T–2	5.091	196.4	143.3
XC6VLX75T–3	4.595	217.6	158.8
XC7A100T–1	7.069	141.4	103.2
XC7A100T–2	5.825	171.6	125.2
XC7A100T–3	5.197	192.4	140.4
XC7K70T–1	4.669	214.1	156.2
XC7K70T–2	3.836	260.6	190.2
XC7K70T–3	3.572	280.0	204.2

Max–Log–MAP, $I = 5$, $K = 6144$, $L = 32$, $N_p = 8$

Signal Descriptions

C MAP Decoder Constant (0–11)
CLK System Clock
DEC_END Decode End Signal
ERR Estimated Error
F1I LTE external parameter 1 (0 to 255)
F1I = f_1 div 2, f_1 odd.
F2I LTE external parameter 2 (0 to 511)
F2I = f_2 div 2, f_2 even.
FS LTE external parameter select
0 = Select internal parameters
1 = Select external parameters
K Data Length (K = 40–6144)
K[12:3] = K div 8
LIMZ Extrinsic Information Limit (1–127)
M Early Stopping Mode
0 = no early stopping
1 = early stop at odd half iteration
2 = early stop at even half iteration
3 = early stop at any half iteration

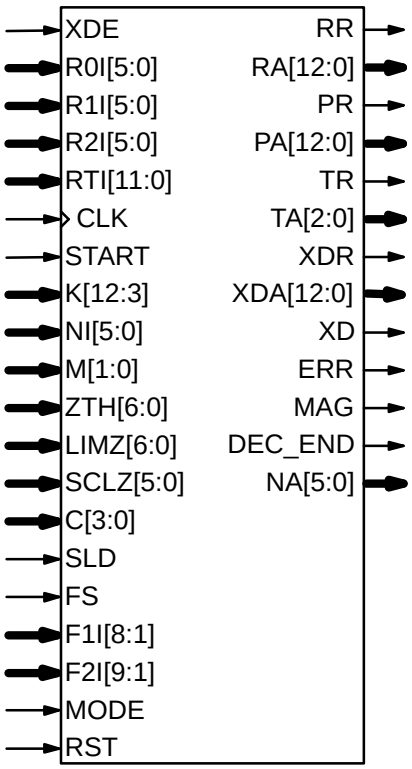

Figure 4: PCD03L1 schematic symbol.

MAG Input Data Magnitude Indicator
MODE Implementation Mode
0 = Max–log–MAP
1 = Log–MAP
NA Half Iteration Number (0–63)
NI Number of Half Iterations (0–63)
NI = 2I–1 where I is number of iterations
PA Address for R1I and R2I
PR Data Ready for R1I and R2I
RA Address for R0I
RR Data Ready for R0I
RTI Received Tail Data
R0I Received Systematic Data
R1I Received Systematic Data Parity
R2I Received Interleaved Data Parity
RST Synchronous Reset
SCLZ Extrinsic Information Scale (1–32)
SLD Sliding window select
0 = small window (L = 32 for K/N_p > 32 else $L = K/N_p$)
1 = large window (L = 64 for K/N_p > 64 else $L = K/N_p$)
START Decoder Start
TA Address for RTI
TR Data Ready for RTI
XD Decoded Data Byte
XDA Decoded Data Address
XDE Decoded Data Enable
XDR Decoded Data Ready

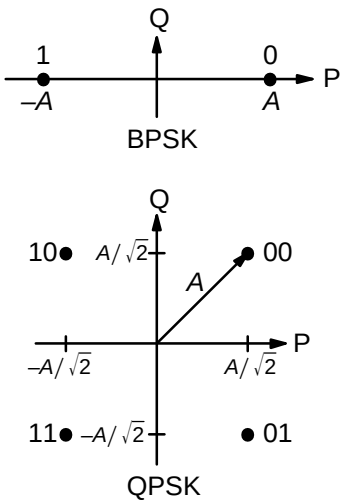

Figure 5: BPSK and QPSK signal sets.

ZTH Early Stopping Threshold (1–127)

The MODE input is used to select various decoder implementations. Note that MODE is a “soft” input and should not be connected to input pins or logic. This input is designed to minimise decoder complexity for the configuration selected.

Note that the required size of the internal interleaver memory is 768x64 for N_p = 8. This is implemented using four 1Kx18 Xilinx BlockRAMs.

Turbo Decoder Parameters

For optimal performance, the maximum a posteriori (MAP) [2] constituent decoder is used which is dependent on the signal to noise ratio (SNR). Unlike other turbo decoders with suboptimum soft–in–soft–in (SISO) decoders, using the MAP (or specifically the log–MAP [3]) algorithm can provide up to 0.5 dB coding gain at low SNRs. Log–MAP operation is enabled when MODE is high.

With binary phase shift keying (BPSK, m = 1) or quadrature phase shift keying (QPSK, m = 2) modulation (see Figure 5) the decoder constant C should be adjusted such that

$$C = A\sigma^2\sqrt{m}/2. \quad (1)$$

where A is the signal amplitude and σ^2 is the normalised noise variance given by

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

E_b/N_0 is the energy per bit to single sided noise density ratio, $R = 1/(n+6n/K)$, K is the data length, and n = 3. C should be rounded down to the nearest integer and limited to be no higher than 11. Max–log–MAP [3] operation occurs when C = 0. Due to quantisation effects, C = 1 is equivalent to C = 0. Thus C = 1 is internally rounded down to C = 0. Also, as C = 2 or 3 give poor performance due

to quantisation effects, these values are internally rounded up to C = 4. Max–Log–MAP operation is also enabled when MODE is low.

For each code (with a particular block size, rate and number of iterations), there will be a minimum E_b/N_0 where the maximum acceptable BER or FER is achieved. The value of C should be chosen for this E_b/N_0. This value of C can be kept constant for all E_b/N_0 values for this code. For higher values of E_b/N_0, there will be neglible degradation in performance, even though C will be higher than optimal [4]. For lower E_b/N_0 values, there could be up to a few tenths of a dB degradation, since C will be lower than optimal. However, this should not have much impact since the BER or FER will already be above the maximum acceptable level anyway.

For fading channels the value of A and σ^2 should be averaged across the block to determine the average value of C. Each received value r_k should then be scaled by $(A\sigma^2)/(A_k\sigma_k^2)$ where A_k and σ_k^2 are the amplitude and normalised variance of r_k. Note that this scaling should be performed for both the log–MAP and max–log–MAP algorithms for optimal performance.

The value of A directly corresponds to the 6–bit signed magnitude inputs (shown in Table 3). The 6–bit inputs have 63 quantisation regions with a central dead zone. 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 15 (which has a range between 15 and 16).

Table 3: Quantisation for R0I[5:0], etc.

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
33	100001	–1.5↔–0.5
34	100010	–2.5↔–1.5
⋮	⋮	⋮
62	111110	–30.5↔–29.5
63	111111	–∞ ↔–30.5

Since most analogue to digital (A/D) converters do not have a central dead zone, a 7–bit A/D should be used and then converted to 6–bit as shown in the table. This allows maximum performance to be achieved.

For signed magnitude inputs a decimal value of 32 has a magnitude of 0 (equivalent to –0). For two's complement inputs, if 32 is input, this will be externally limited to 33 (equivalant to –31).

For input data quantised to less than 6–bits, the data should be mapped into the most significant bit positions of the input, the next bit equal to 1 and the remaining least significant bits tied low. For example, for 3–bit received data R0T[2:0], where R0T[2] is the sign bit, we have R0I[5:3] = R0T[2:0] and R0I[2:0] = 4 in decimal (100 in binary). For punctured input data, all bits must be zero, e.g., R1I[5:0] = 0.

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

Example 1: Rate 1/3 BPSK code operating at E_b/N_0 = 0.3 dB. From (2) we have σ^2 = 1.39988. Assuming A = 9 we have from (1) that C = 6.29946 which is rounded down to 6.

Figure 6 gives a simplified block diagram of the PCD03L8 eight state turbo decoder. The MAP decoders are designed to accept input and output data in forward or reversed blocks of L. This reduces the MAP decoder delay to 2L (instead of 4L as in continious MAP decoders).

The last forward and reverse state metrics (SM) are also stored in two separate registers for odd and even half iterations. The stored forward SMs are used as the starting states in the following MAP decoder at the next full iteration. Simlarly, the stored reverse SMs are used by the previous MAP decoder in the next full iteration. This is neccesary as for small K, the sliding window size is very small (L = 5 for K = 40 and N_p = 8), implying that the SMs do not become reliable after starting from all zero values. By using the SMs from the previous iteration as the starting values, this gives a more reliable decoded output.

The sliding window algorithm used depends on L. For $L = K/N_p$, Figure 7 shows how the forward and reverse state metrics (SM) are calculated. The horizontal axis shows decoder time. The vertical axis the received symbol index. An arrow going up shows forward SMs for L symbols. An arrow going down shows reverse SMs for L symbols. A horizontal dashed arrow indicates SMs being passed between iterations. Forward SMs are indicated by an F and reverse SMs by R. Forward SMs that have been reversed in time are indicated by E. Decoded data is output when R and E are used together.

We see that in this case the RAMs are read in forward blocks of L. Decoded data is written in reversed blocks of L. Also, the SMs are allowed to

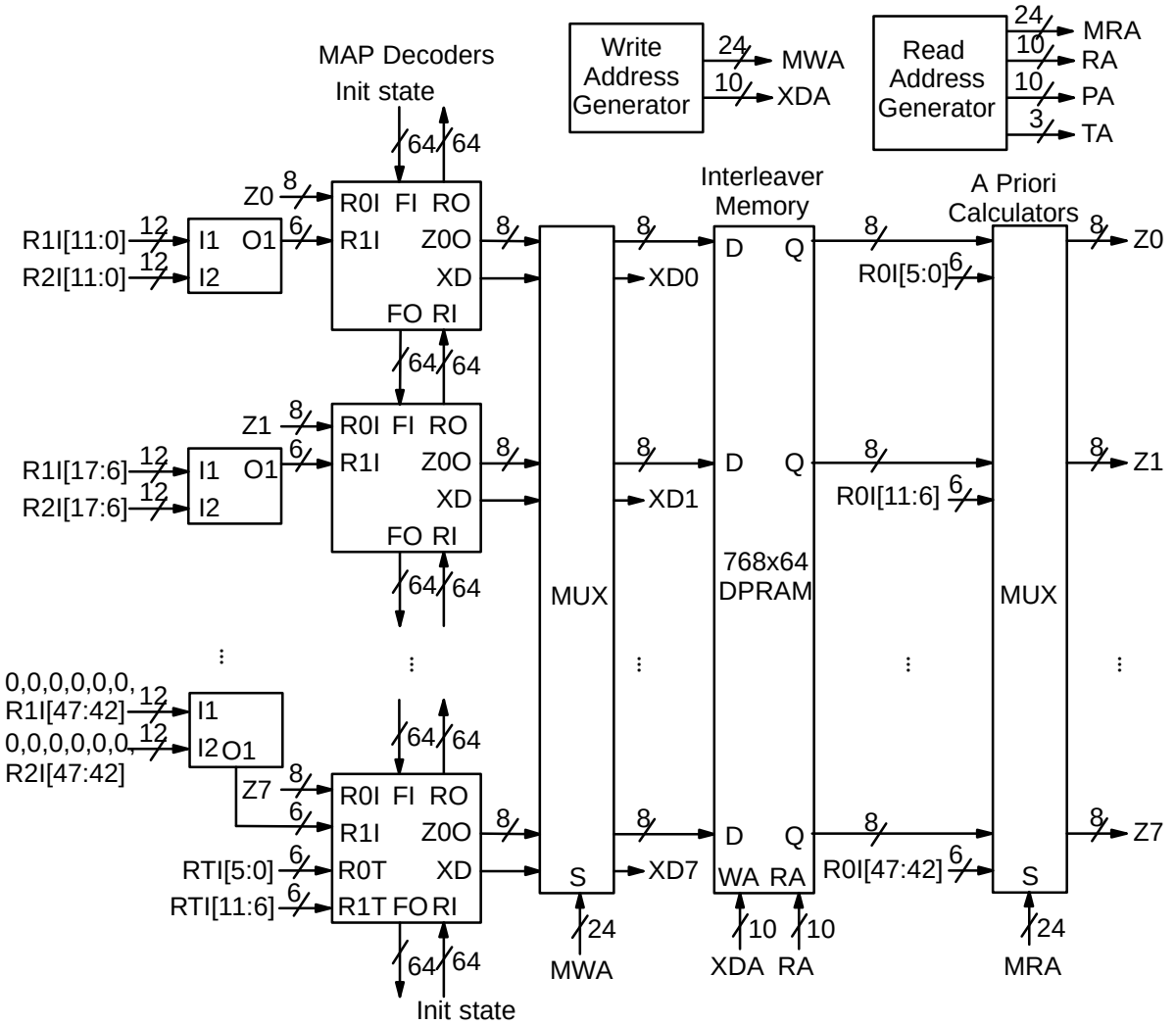

Figure 6: Simplified block diagram of PCD03L8 eight state turbo decoder.

settle for L symbols before being used. As L can be very small, e.g., $L = 5$ for $K = 40$ and $N_p = 8$, this is not sufficient to obtain reliable SMs. Thus, we pass both forward and reverse SMs between iterations to help improve the reliability of the SMs.

Figures 8 to 10 show the timing diagram for various cases where $K/N_p > L$. Only the trellis diagrams for the first two MAP decoders are shown. Unlike the previous case, we read the SMs in reverse blocks of L and output them in reverse order. We pass forward SMs between iterations so that the starting forward SMs become reliable. This is indicated by the dashed arrow going backwards in time. As the reverse state metrics are reliable when used to calculate the extrinsic information there is no need to pass reverse SMs between iterations.

However, for $K/N_p > 2L$ the final SMs of the third block, for example R3 in Figure 9, are stored and then passed to the previous MAP decoder. This is indicated by the dashed arrow going forwards in time. For $L < K/N_p \leq 2L$ (Figure 8), the reverse SMs for the second reverse SM calculator are passed directly between MAP decoders.

For $L < K/N_p \leq 3L$ (Figures 8 and 9), the first reverse SM calculator initially uses input data from the next MAP decoder for the last input block in time. This is why the R1I and R2I inputs shown in Figure 6 have an input width twice that of a single sample of six bits. This method avoids additional multiplexers to select the first reverse SMs from the next MAP decoder. However, this technique can not be used for $K/N_p > 3L$ as the old extrinsic information will have been overwritten with new data. Instead, we use the previously stored reverse SMs as shown by the first forward arrow in Figure 10.

For $K/N_p > 4L$ the timing diagrams are similar to that for $3L < K/N_p \leq 4L$ (Figure 10). In all cases, we see that decoding time is equal to K/N_p+2L. We also need to add an additional 10 clock cycles for pipeline delay, to give a decoding time of

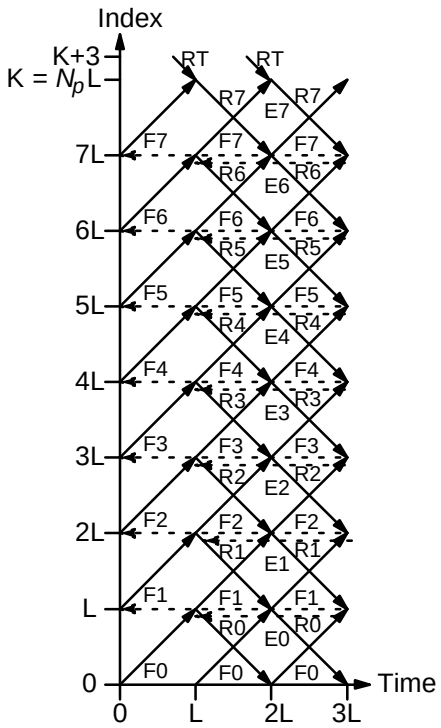

Figure 7: Timing diagram for $L = K/N_p$.

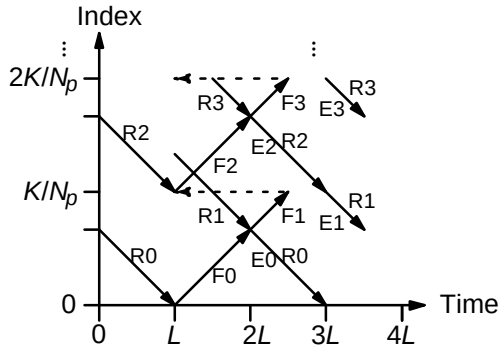

Figure 8: Timing diagram for $L < K/N_p \leq 2L$.

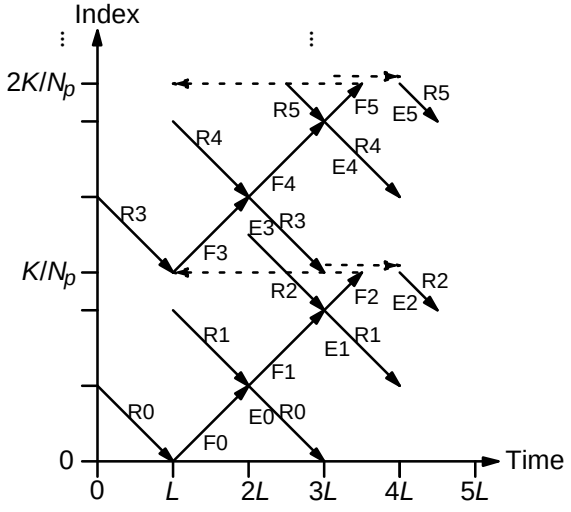

Figure 9: Timing diagram for $2L < K/N_p \leq 3L$.

K/N_p+2L+10 clock cycles for each half iteration. Two of the 10 clock cycles are for calculating the

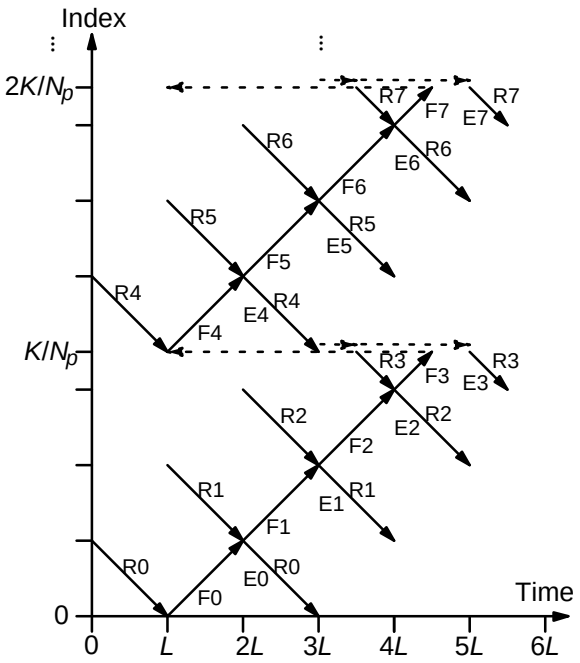

Figure 10: Timing diagram for $3L < K/N_p \leq 4L$.

a priori information and eight clock cycles for the MAP decoder delay. One additional clock cycle is used to start the turbo decoder.

The number of turbo decoder half–iterations is given by NI, ranging from 0 to 63. $NI = 2I–1$ where I is the number of iterations. This is equivalent to 0.5 to 32 iterations. The decoder initially starts at half iteration NA = 0, increasing by one until NI is reached or an earlier time if early stopping is enabled. The NA output can be used to select LIMZ and SCLZ values, which is useful for max–log–MAP decoding.

The turbo decoder speed f_d is given by

$$f_d = \frac{F_d}{(NI + 1)(1/N_p + (2L + 10)/K)} \qquad (3)$$

where F_d is the CLK frequency, L is the MAP decoder sliding window length and N_p the number of parallel MAP decoders. Table 4 gives the value of L depending on K, N_p and SLD. SLD = 0 can be used to increase decoder speed, while SLD = 1 should be used for high puncturing rates to increase performance.

Table 4: Sliding window length

K		L	
min	max	SLD=0	SLD=1
40	256	K/N_p = 5..32	K/N_p = 5..32
264	512	32	K/N_p = 33..64
520	6144	32	64

For example, if F_d = 100 MHz, I = 5 (NI = 9) and N_p = 8, the decoder speed ranges from 16 Mbit/s

for K = 40 and L = 5 to 72.9 Mbit/s for K = 6144 and L = 32.

An important parameter is LIMZ, the limit factors for the extrinsic information. Extrinsic information is the "correction" term that the MAP decoder determines from the received data and *a priori* information. It is used used as *a priori* information for the next MAP decoding or half iteration. By limiting the correction term, we can prevent the decoder from making decisions too early, which improves decoder performance.

The limit factor LIMZ should vary between 1 and 127, although we recommend that 96 be used.

Another parameter that can used to adjust decoder performance is SCLZ which ranges from 1 to 32. The extrinsic information is scaled by SCLZ/32 followed by rounding to the nearest integer. Thus, when SCLZ = 32, no scaling is performed. For log–MAP decoding we recommend SCLZ = 29. For max–log–MAP decoding we recommend SCLZ = 23. The NA output can be used to adjust LIMZ and SCLZ with the number of iterations for optimum performance.

There are four decoder operation modes given by M. Mode M = 0 decodes a received block with a fixed number of iterations (given by NI). Modes 1 to 3 are various early stopping algorithms. Early stopping is used to stop the decoder from iterating further once it has estimated there are zero errors in the block. Mode 1 will stop decoding after an odd number of half–iterations. Mode 2 will stop decoding after an even number of half iterations. Mode 3 will stop after either an odd or even number of half iterations. Further details are given in the next section.

Turbo Decoder Operation

After the START signal is sent, the decoder will read the received data at the CLK speed. It is assumed that the received data is stored in three synchronous read RAMs of size (K/N_p)x$(6N_p)$ and one synchronous read 6x12 RAM for the tail data. The RAMs for R1I and R2I (using PA and PR) can be combined into a single $(K/8)$x$(12N_p)$ RAM if desired.

For input RiI[6j+5:6j][k], i = 0 to 2, j = 0 to N_p–1, and k = 0 to K/N_p–1, the input data corresponds to code symbol $3(K/N_p)j+3k+i$. For input RTI[6j+5:6j][k], j = 0 to 1 and k = 0 to 5, the input data corresponds to code symbol $3K+2k+j$. The read address for input R0I is given by RA, for input R1I and R2I is given by PA and for input RTI is given by TA.

The received data ready signal RR, PR and TR goes high to indicated the data to be read from the address given by RA, PA and TA, respectively. Table 5 illustrates which data is stored for address 0 to K/N_p–1 for the main data, 0 to 2 for the first tail, and 3 to 5 for the second tail. The entries for the table indicate which encoded data output is selected, X, Y1 for the first encoder and X′, Y1′ for the second encoder. The code polynomials are $g^0(D) = 1+D^2+D^3$ (13 in octal) and $g^1(D) = 1+D+D^3$ (15).

Table 5: Input data format

Rate	Data	Main	Tail 1	Tail 2
1/3	R0I[6N_p–1:0]	X		
	R1I[6N_p–1:0]	Y1		
	R2I[6N_p–1:0]	Y1′		
	RTI[5:0]		X	X′
	RTI[11:6]		Y1	Y1′

The decoder then iteratively decodes the received data for NI+1 half iterations, rereading the received data for each half iteration for K/8+L CLK cycles. The signal RR and PR goes high for K/8+L CLK cycles while data is being output. As the last input RAM has the tail, a separate read address TA and read ready TR signals are used. TR goes high for 3 CLK cycles to read the tail.

Figure 11 illustrates the decoder timing where the data is input during the first half iteration. The input $\boldsymbol{R}_k^i$, for i = 0 to 2 and k = 0 to K/N_p–1 corresponds to input data RiI[6N_p–1:0][k]. The input $\boldsymbol{R}_k^T$ for k = 0 to 5 corresponds to RTI[11:0][k].

Note that while DEC_END is low (decoding is being performed), the START signal is ignored, except for the last clock cycle before DEC_END goes high. A synchronous reset is also provided. All flip flops in the turbo decoder are reset during a low to high transition of CLK while RST is high.

The decoded data is output during the last half–iteration on XD[N_p–1:0]. That is, decoded data is output N_p–bits every CLK cycle. The signal XDR goes high for K/N_p CLK cycles while the block is output. If NI is even (odd half iterations), the block is output in reverse block sequential order. To dereverse the decoded data, the output XDA[9:0] needs to be used as the write address to a buffer RAM.

For NI odd (even half iterations), the block is output in reverse block interleaved order. To dereverse and deinterleave the block, the output XDA[9:0] is used as the write address to a buffer RAM.

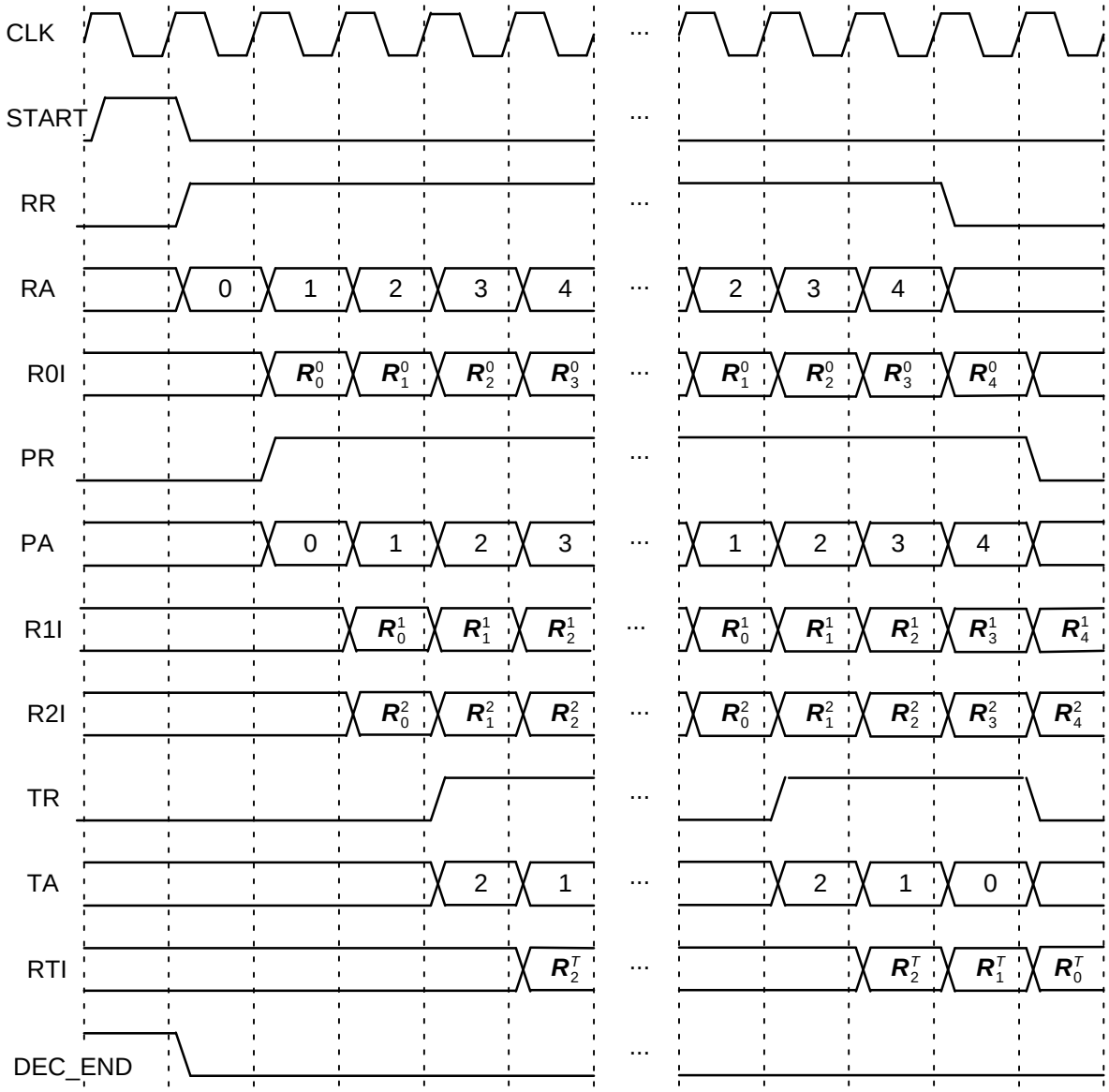

Figure 11: Turbo Decoder Input Timing (K = 40, N_p = 8, L = 5, first half iteration).

The bus ERR[N_p–1:0] is a channel error estimator output. It is the exclusive OR of XD[N_p–1:0] and the sign bits of R0I[6N_p–1:0], i.e., bit R0I[6j+5] for j = 0 to N_p–1.

The bus MAG[N_p–1:0] is the input data magnitude indicator. For R0I[6j+5:6j] for j = 0 to N_p–1, MAG[j] is zero if R0I[6j+4:6j] = 0, otherwise MAG[j] is one. For example if R0I[5:0] = 16 then MAG[0] = 1. If R0I[5:0] = 0 or R0I[5:0] = 32 then MAG[0] = 0.

The MAG[N_p–1:0] outputs can be used to erase ERR[N_p–1:0] outputs when input data R0I[6j+4:6j] is erased. This avoids counting errors where the input sign is unknown. For example, say non–erased data has a minimum input magnitude of one. If R0I[5:0] has its magnitude erased, i.e., R0I[4:0] is zero, then by using the operation ERR[0] AND MAG[0], we can delete this estimated error.

The DEC_END signal is low during decoding. At the end of decoding, DEC_END goes high. Figure 12 illustrates the decoder timing where data is output on the last half iteration. After startup, the maximum number of clock cycles for decoding is (NI+1)(K/N_p+2L+10).

During the last half iteration the decoded and error data are stored into the interleaver memory. Once decoding has been completed, the input XDE can be used to sequentially clock the decoded and error data from from the interleaver memory (regardless of the number of iterations). XDE is disabled while the decoder is iterating. Figure 13 shows the decoder timing when XDE is used.

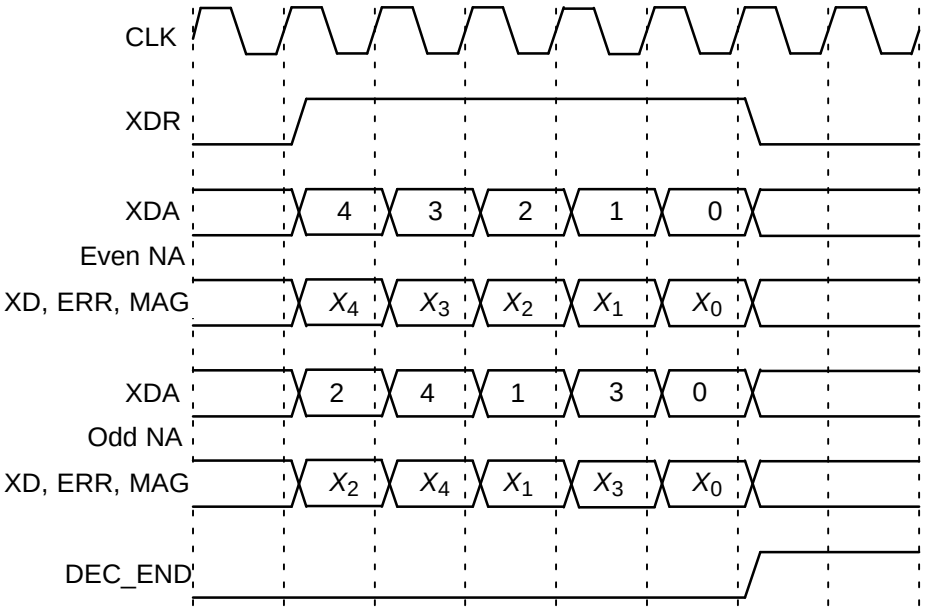

Figure 12: Turbo Decoder Output Timing (K = 40, N_p = 8, L = 5).

The early stopping algorithm uses the magnitude of the extrinsic information to determine when to stop. As the decoder iterates, the magnitudes generally increases in value as the decoder becomes more confident in its decision. By comparing the smallest magnitude of a block with threshold ZTH, we can decide when to stop. If the smallest magnitude is greater than ZTH, i.e., not equal or less than ZTH, the decoder will stop iterating if early stopping has been enabled.

Since the last half iteration is used to store the decoded data into the interleaver memory, the decoder performs an extra half iteration once the threshold has been exceeded.

Increasing ZTH will increase the average number of iterations and decrease the BER. Decreasing ZTH will decrease the average number of iterations and increase the BER. In general, higher values of SNR will decrease the number of iterations. A value of ZTH = 23 was found to give a good trade off between the average number of iterations and BER performance.

For high SNR operation early stopping can lead to significantly reduced power consumption, since most blocks will be decoded in one or two iterations.

Simulation Software

Free software for simulating the PCD03L8 turbo decoder in additive white Gaussian noise (AWGN) or with external data is available by sending an email to info@sworld.com.au with “pcd03l8sim request” in the subject header. The software uses an exact functional simulation of

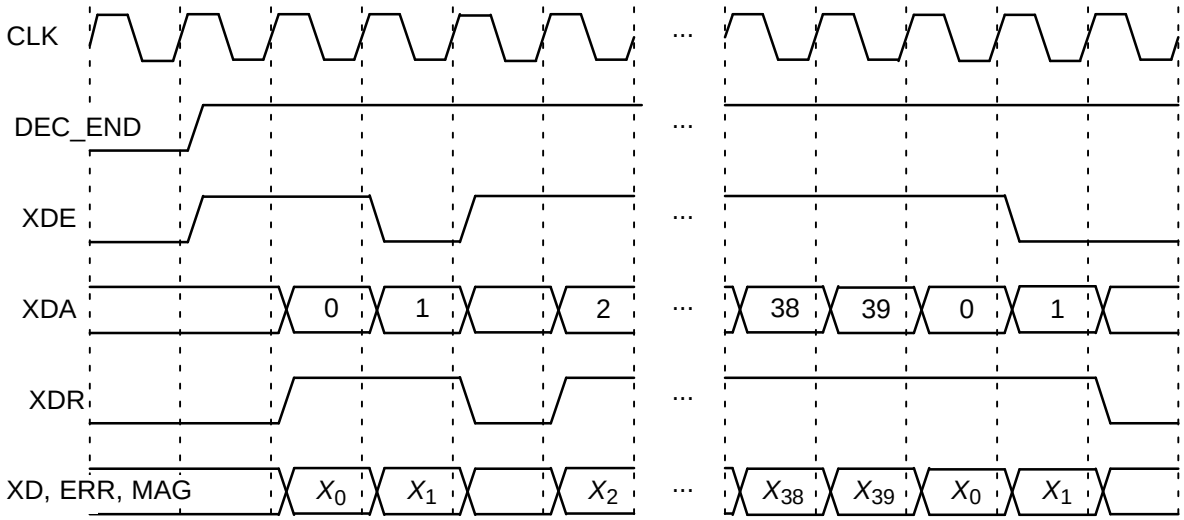

Figure 13: XDE Timing (K = 320).

the PCD03L8 turbo decoder, including all quantisation and limiting effects.

After unzipping pcd03l8sim.zip, there should be pcd03l8sim.exe and code.txt. The file code.txt contains the parameters for running pcd03l8sim. These parameters are

`kt`	No. of data bits per punctured symbol (1 or greater)
`nt`	No. of coded bits per punctured symbol (2 or 3 if `kt`=1 or `kt`+1 if `kt` > 1)
`m`	Encoder memory (3)
`g0`	Divisor polynomial in octal (13)
`g1`	Parity polynomial in octal (15)
`q`	Number of quantisation bits (1 to 6)
`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 (0.37 for rate 1/3)
`ferrmax`	Number of frame errors to count
`Pfmin`	Minimum frame error rate (FER)
`Pbmin`	Minimum bit error rate (BER)
`NI`	Number of half iterations–1 (0 to 63)
`SLD`	MAP decoder delay select (0 or 1)
`LIMZ`	Extrinsic information limit (1 to 127)
`SCLZ`	Extrinsic information scale (1 to 32)
`M`	Stopping mode (0 to 4)
`ZTH`	Extrinsic info. threshold (0 to 127)
`NP`	Number of MAP decoders (1, 2, 4 or 8)
`K`	Block length (40 to 6144)
`FS`	LTE interleaver parameter select (0 or 1)
`f1`	LTE interleaver parameter 1 (1 to 511)
`f2`	LTE interleaver parameter 2 (0 to 1022)
`LOGMAP`	Log–MAP decoding (MODE, 0 or 1)
`enter_C`	Enter external C (y or n)
`C`	*C* (0 to 11)
`state`	State file (0 to 2)
`s1`	Seed 1 (1 to 2147483562)
`s2`	Seed 2 (1 to 2147483398)
`out_screen`	Output data to screen (y or n)
`read_x`	Use external information data (y or n)
`read_r`	Use external received data (y or n)
`out_dir`	Output directory
`PP`	Puncture Period (1 to 255)
`BC`	Bits per Character (1 to 5, 1 = binary, 3 = octal, 4 = hexadecimal)
`P0`	Systematic Puncture Pattern
`P1`	First Parity Puncture Pattern
`P2`	Second Parity Puncture Pattern

The parameter `optC` is used to determine the "optimum" values of *A* and *C*. The "optimum" value of *A* is

$$A = \frac{\text{optC}(2^{q-1} - 1)}{\text{mag}(\sigma)} \quad (4)$$

where 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 have

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

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

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

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.

For the "optimum" *A*, we round the value of *C* given by (1) down to the nearest integer. If LOGMAP = MODE = 0 then *C* is forced to 0. If LOGMAP = 1, *C* is limited to a maximum value of 11. If *C* = 1, *C* is rounded down to 0. If *C* = 2 or 3, *C* is rounded up to 4. An external value of *C* can be input by setting `enter_C` to `y`.

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, for example, file k512.dat, where *K* = 512. Only frames that are in error are stored in the output file. 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`), the average BER (`Pb`) and the average FER (`Pf`). Following this, the number of iterations, `na`, `berr`, `ferr`, `Pb`, and `Pf` are given for each half iteration.

The following file was used to give the simulation results shown in Figure 14 for *K* = 512, N_p = 8 and log–MAP decoding. Auto–stopping was used with up to 10 iterations. When iterating is stopped early, the `nasum` (`2*num*na`), `berr` and `ferr` results at stopping are copied for each half iteration to the maximum iteration number. Figure 15 shows the average number of iterations with E_b/N_0.

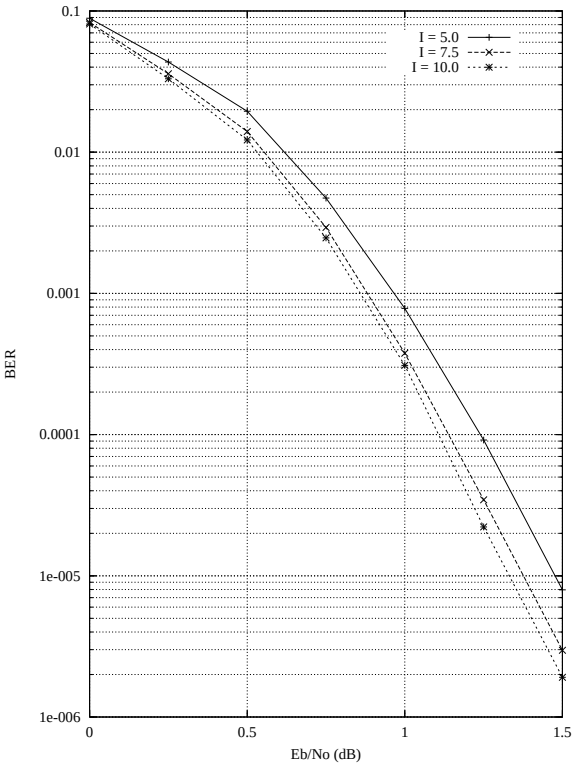

Figure 14: BER performance with K = 512, N_p = 8, log–MAP and auto–stopping.

```
{kt nt m g0 g1}
 1  3  3  13 15
{q  EbNomin EbNomax EbNoinc optC}
 6  0.00     1.50    0.25    0.37
{ferrmax Pfmin Pbmin}
 128      1e-99 1e-5
{NI  SLD  LIMZ  SCLZ  M  ZTH}
 19  0    96    29    3  23
{NP  K    FS  f1  f2}
 8   512   0   3  10
{LOGMAP  enter_C  C}
 1       y        6
{state s1    s2     out_screen}
 0     12345 67890  y
{read_x  read_r  out_dir  in_dir}
 n       n       output   input
{PP BC P0 P1 P2}
  1  1  1  1  1
```

Figure 16 shows the performance of the turbo decoder for various block sizes, log–MAP and max–log–MAP decoding. For reference, the performance of the PCD03V turbo decoder with one MAP decoder and Genie aided early stopping is shown. The Genie can be employed by selecting M = 4 in code.txt (this option is not available in the core).

For K = 40, log–MAP and early stopping with ZTH = 23, the PCD03L8 turbo decoder loses only 0.08 dB at a BER of 10^{-5} compared to the PCD03V turbo decoder with genie aided early stopping. This is despite the sliding window length L being only equal to 5 for K = 40. Max–log–MAP and K = 40 shows a performance loss of 0.23 dB at high BER's and 0.09 dB at low BER's.

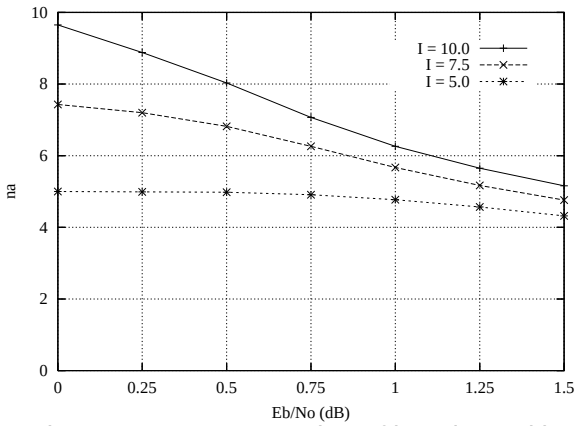

Figure 15: Average number of iterations with K = 512, N_p = 8, log–MAP and auto–stopping.

For K = 256 and longer lengths, L increases to at least 32. For log–MAP the performance of Genie aided PCD03V compared to PCD03L8 with early stopping and ZTH = 23 are virtually identical.

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:

1) Copy the state files state1.dat and state2.dat. This ensures you can restart the program if a mistake is made in configuring code03l8.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_512.dat` in directory `input` (each line contains one bit of data). The encoded stream `y_(K).dat` will be output to the directory given by `out_dir`, e.g., `y_512.dat` to directory `output`.

To decode data, place the received block of data in file `r_(K).dat` in directory `in_dir` and set

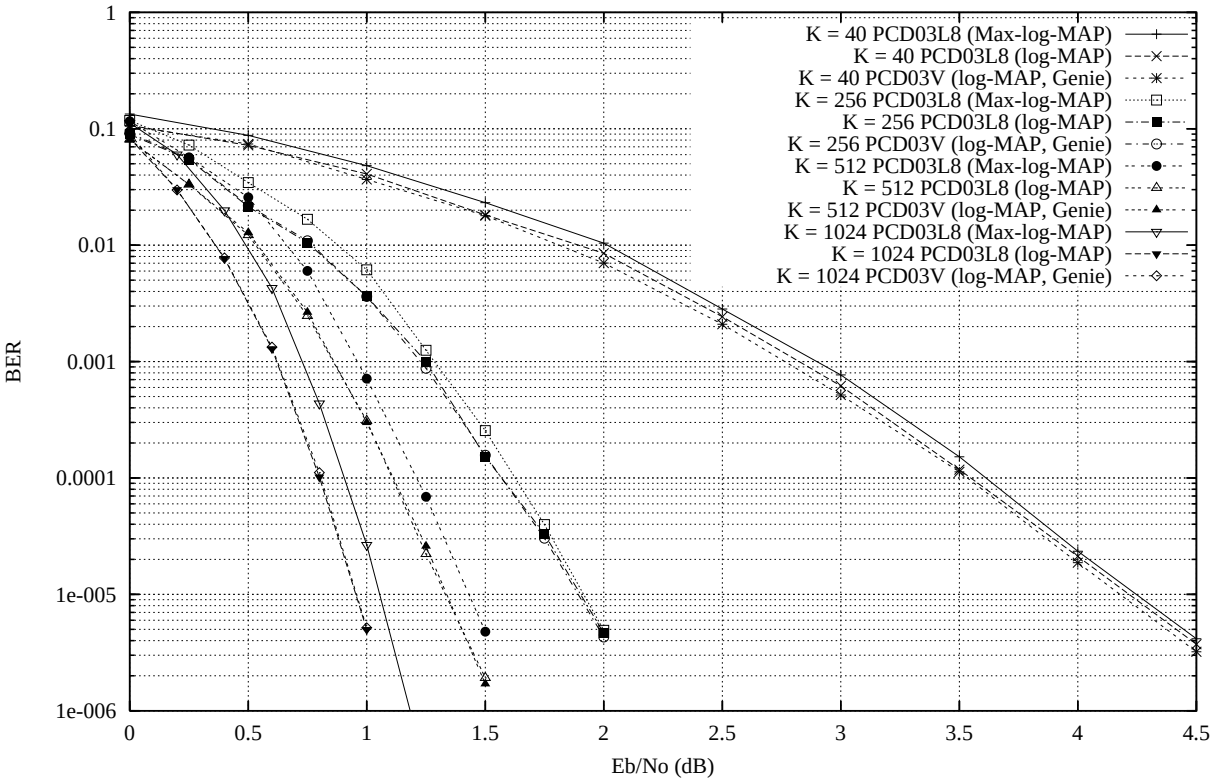

Figure 16: Performance with various block sizes, auto–stopping (Genie and ZTH = 23), log–MAP (SCLZ=29) and max–log–MAP (SCLZ=23).

`read_r` to `y`. The decoded data is output to `xd_(K).dat` in directory `out_dir`. For non–punctured data, `r_(K).dat` has in each line R[i,j], i = 0 to nt–1, j = 0 to *K*–1 and i = 0 to n–1, j = *K* to *K*+2*m*–1, e.g., for kt = 1 and nt = 3 the first four lines would be

–31
1
–25
–31

where R[0,0] = –31, R[1,0] = 1, R[2,0] = –25 and R[0,1] = –31. The input data is of the form
R[i,j] = A*(1–2*Y[i,j]+N[i,j])

where A is the signal amplitude, Y[i,j] is the coded bit, and N[i,j] is white Gaussian noise with zero mean and normalised variance σ^2. The magnitude of R[i,j] should be rounded to the nearest integer and be no greater than 2^{q-1}–1. If `read_r` = `y`, then *C* is externally input via `c`.

There are two ways to simulate punctured data. In both cases the tail bits are not punctured.

If `PP` = 1, then puncturing can be performed by modifying `kt` and `nt`. Rate *k*/(*k*+1) can be simulated by letting `kt` = *k* and `nt` = *k*+1. The puncturing period is 2*k*. The systematic bits are not punctured, the first parity has all bits punctured except bit positions 2*ki*+*k* and the second parity has all bits punctured except bit positions 2*ki*+*k*–1, where *i* ranges from 0 to *K*/(2*k*)–1. For example, for rate 2/3, the puncturing patterns are 1111, 0010 and 0100 for the systematic, first parity and second parity, respectively.

If `PP` > 1, `kt` = 1 and `nt`= 3, the puncturing pattern can be input externally. The puncturing period is given by `PP`. The puncturing patterns are given by `P0`, `P1` and `P2` for the systematic, first parity and second parity, respectively. Encoding can be binary, octal or hexadecimal using `BC` = 1, 3 or 4, respectively. For example `PP` = 4, `BC` = 3, `P0` = 17, `P1` = 02 and `P1` = 04 gives the same puncturing pattern as the previous example.

Due to the tail, the nominal code rate will be slightly less than kt/nt. The exact code rate is calculated by the software and provided as one of the outputs. With puncturing, SLD = 1 is recommended for best performance. For external input data `r_(K).dat`, if puncturing is selected, the software will depuncture the data. If depuncturing is performed in `r_(K).dat`, then `kt` = 1, `nt` = 3 and `PP` = 1 need to be selected.

Note that the core itself does not perform any depuncturing. This needs to be performed external to the decoder.

Ordering Information

SW–PCD03L8–SOS (SignOnce Site License)
SW–PCD03L8–SOP (SignOnce Project License)
SW–PCD03L8–VHD (VHDL ASIC License)

All licenses include Xilinx VHDL cores. All licenses allows unlimited instantiations.

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] Third Generation Partnership Project (3GPP), "Evolved universal terrestrial radio access (E–UTRA); Multiplexing and channel coding," 3GPP TS 36.212 V8.1.0 Release 8, Nov. 2007.

[2] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT–20, pp. 284–287, Mar. 1974.

[3] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub–optimal MAP decoding algorithms operating in the log domain," *ICC'95*, Seattle, WA, USA, pp. 1009–1013, June 1995.

[4] M. C. Reed and J. A. Asenstorfer, "A novel variance estimator for turbo–code decoding," *Int. Conf. on Telecommun.*, Melbourne, Australia, pp. 173–178, Apr. 1997.

Small World Communications does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its copyrights or any rights of others. *Small World Communications* reserves the right to make changes, at any time, in order to improve performance, function or design and to supply the best product possible. *Small World Communications* will not assume responsibility for the use of any circuitry described herein. *Small World Communications* does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. *Small World Communications* assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. *Small World Communications* will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Supply of this IP core does not convey a license nor imply any right to use turbo code patents owned by France Telecom, GET or TDF. Please contact France Telecom for information about turbo codes licensing program at the following address: France Telecom R&D – VAT/Turbocodes, 38 rue du Général Leclerc, 92794 Issy Moulineaux Cedex 9, France.

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

Version History

- 0.00 15 April 2010. Preliminary product specification.
- 0.01 21 April 2010. Added sliding window length table.
- 0.02 26 April 2010. Updated simplified block diagram and turbo decoder input timing figures. Corrected timing diagrams for $L = K/8$ and $K/32$.
- 0.03 1 June 2010. Changed input data format from eight 18–bit inputs to three 48–bit inputs. Added optional interleaver parameters. Added tail address address and ready signal. Updated turbo decoder input timing figure.
- 0.04 27 July 2010. Reduced maximum number of iterations from 128 to 32. Added XDE input. Changed parity address and ready signal from R2A[9:0] and R2R to PA[9:0] and PR. Changed tail address and ready signal from RTA[2:0] and RTR to TA[2:0] and TR. Updated simplified block diagram. Reduced number of clock cycles by one.
- 0.09 8 October 2010. Added MAG[7:0] output. Updated performance. Added Virtex–6 and Spartan–6 performance. Updated Virtex–4 and Virtex–5 complexity. Updated turbo decoder input timing. Added performance with various block sizes.
- 1.00 9 October 2010. First official release.
- 1.02 26 November 2010. Improved short block length BER performance. Updated Virtex–6 speed. Updated Virtex–4 and Virtex–5 complexity. Updated and improved timing diagrams. Added MAG to turbo decoder output timing and

XDE timing figures. Updated performance with various block sizes.
- 1.03 28 December 2010. Added truncation information for extrinsic information scaling. Changed parameter file for pcd03l8sim.exe from pcd03l8.txt to code03l8.txt.
- 1.04 13 March 2014. Added puncturing option for simulation software. Input data is now one line per value.
- 1.05 27 February 2015. Minor corrections and updates.
- 1.06 16 November 2015. Added PCD03L4, PCD03L2 and PCD03L1 cores. Updated PCD03L8 Kintex–7 complexity and speed. Changed parameter file for pcd03l8sim.exe from code03l8.txt to code.txt.
- 1.07 27 November 2015. Updated complexity. Added Virtex–5, Virtex–6 and Artix–7 performance.
- 1.08 1 December 2015. Changed data length input to K[12:3] for PCD03L1, PCD03L2 and PCD03L4. Updated K = 512 BER and NA performance curves. Updated complexity and sliding window length tables.
- 1.09 25 April 2016. Updated decoder complexity. Deleted EDIF cores.
- 1.10 13 May 2016. Updated decoder performance.
- 1.11 3 August 2016. Added external puncturing patterns for BER simulation.
- 1.12 3 October 2016. New sliding window algorithm. Updated decoder complexity, speed and performance.
- 1.13 7 October 2016. Updated decoder complexity.