Application Report
SPRA749B - August 2006

Using TMS320C6416 Coprocessors: Turbo Coprocessor (TCP)

Chad Courtney
Digital Signal Processing Solutions

ABSTRACT

The turbo coprocessor (TCP) is a programmable peripheral for decoding IS2000/3GPP turbo codes, that are integrated into the Texas Instruments (TI™) TMS320C6416 digital signal processor. The TCP is controlled via memory-mapped control registers and data buffers. Control registers can be accessed directly by the CPU, whereas data buffers are typically accessed using the EDMA controller. This application report describes the relationship between the theory of turbo decoding and TCP implementation, outlines TCP programming procedures, and provides examples that demonstrate how to program TCPs for typical 3GPP/IS2000 parameters.

Contents

List of Figures

List of Tables

1 Introduction

The turbo coprocessor (TCP) is a programmable peripheral for decoding of IS2000/3GPP turbo codes, integrated into TI's TMS320C6416 DSP device. The coprocessor operates either as a complete turbo decoder including the iterative structure (standalone processing mode), or it can operate as a single maximum A posterior (MAP) decoder (shared processing mode). In the standalone processing mode, the inputs into the TCP are channel-soft decisions for systematic and parity bits; the outputs are hard decisions. In the shared processing mode, the inputs are channel-soft decisions for systematic and parity bits and apriori information for systematic bits, and the outputs are extrinsic information for systematic bits.

The TCP programmable parameters are:

- Code rate (1/2, 1/3 or 1/4)
- Frame length
- Maximum number of iterations
- Threshold for early stopping criterion

2 Background on Turbo Decoding Algorithm

2.1 Turbo Encoding

Turbo code is a parallel or serial concatenated convolutional code constructed from low-complexity recursive convolutional (constituent) codes. We are only concerned with parallel concatenated convolutional codes (**PCCC**), which were introduced in [1]. An example of such a code is shown in Figure 1. The basic convolutional codes which are concatenated in this scheme are usually two identical recursive codes, with a relatively small number of states (i.e., 8-states for the example in Figure 1). The input to the top encoder, labeled "*Encoder 1,*" is information sequence **U,** and the output are parity sequences **C0** and**C1**. The input to the bottom encoder, labeled "*Encoder 2,*" is block-interleaved version of the original information sequence, denoted **U',** and the output are parity sequences **C0'** and **C1'**. The output of the overall encoder is obtained through puncturing and multiplexing of systematic sequences **U** and **U'** and coded sequences **C0**,**C1**,**C0'** and **C1'**. It usually consists of the systematic component **U**, and two parity components **C** and **C',** where **C** is obtained by puncturing and multiplexing **C0** and **C1,** and **C'** is obtained by puncturing and multiplexing **C0'** and **C1'**.

As an example, rate 1/3 turbo code is obtained when the overall output consists of multiplexed sequences **U**, **C0** and **C0'**.

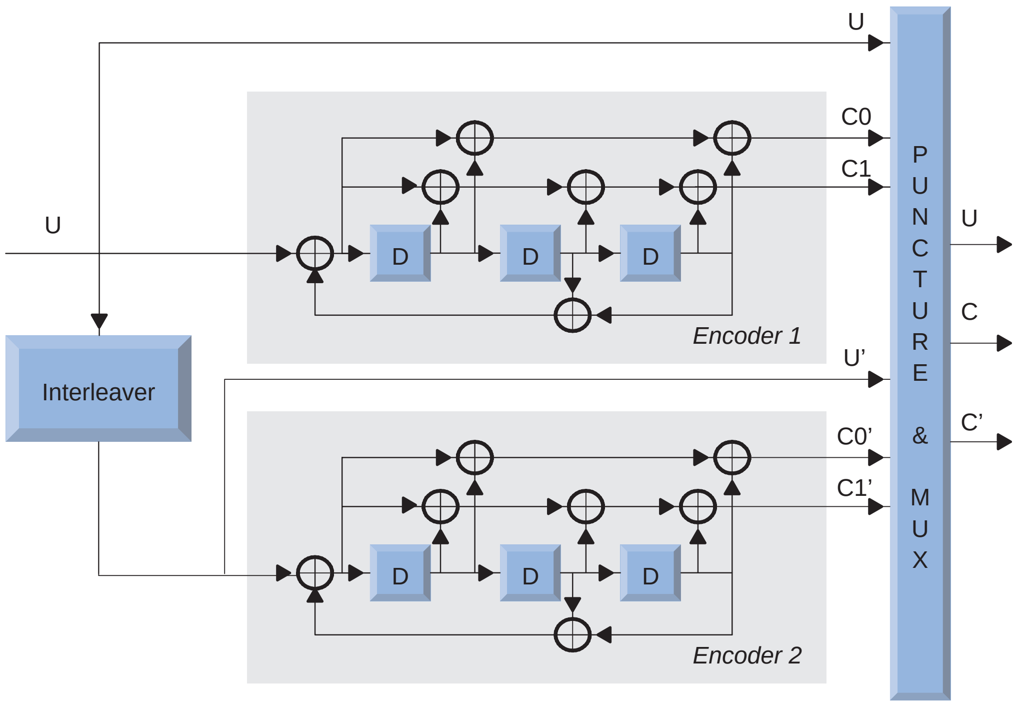

Figure 1. Turbo Code With Recursive 8-State Encoders (15/13, 17/13)

Turbo encoding is performed on blocks of information bits of length N. To facilitate the decoding process, the encoders are in state 0 at the beginning of the frame, and they are also brought to state 0 at the end of the frame. This is achieved by appending tail bits after the information bits. Since the encoder is recursive, the bit sequence which brings the encoder into state zero is not necessarily a zero sequence. Instead, it depends on the state of the encoder after the information bits have been encoded.

2.2 Turbo Decoding

The decoding of turbo codes is based on an iterative structure constructed from two MAP decoders, one for each constituent encoder. A high-level block diagram of a turbo decoder is shown in Figure 2.

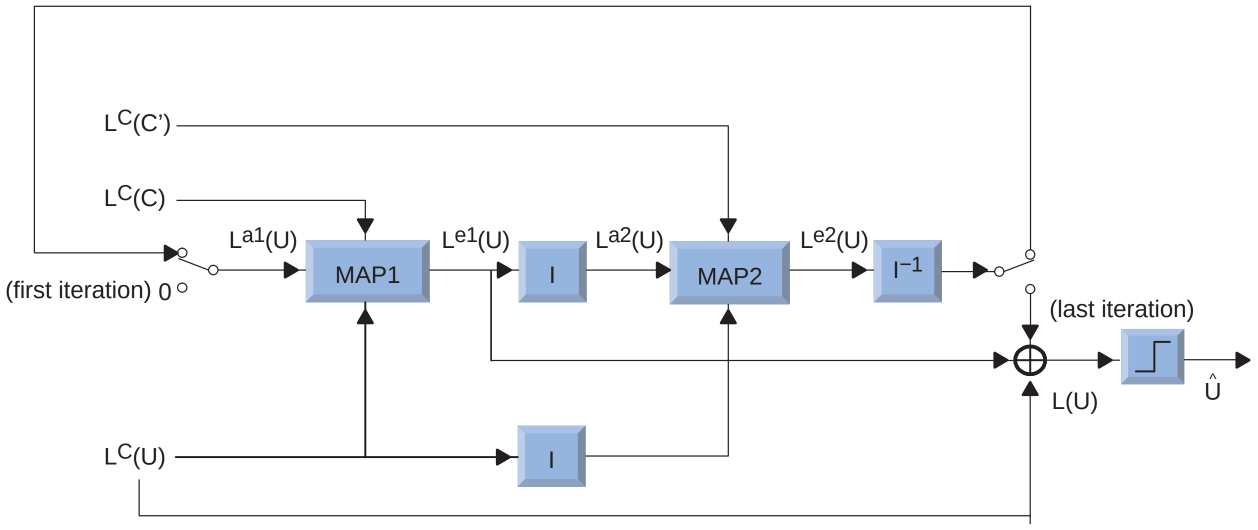

Figure 2. Turbo Decoder

The decoder operates in the log-likelihood ratio domain, i.e., instead of operating on a priori, a posteriori probabilities and channel likelihoods, it operates on the log-likelihood ratio (LLR) of these quantities, defined as:

$$L(u) = \log \frac{\Pr(u = 1)}{\Pr(u = 0)}$$

Each MAP decoder has three inputs: a prior LLR for systematic bits $\mathbf{L^a(U)}$, channel LLR for systematic bits $\mathbf{L^c(U)}$ and channel LLR for parity bits $\mathbf{L^c(C)}$. The computation of channel LLRs is discussed in section 2.2.1.

The output of each MAP decoder is called “extrinsic LLR” and is denoted $\mathbf{L^e(U)}$. This quantity will be described in section 2.2.2.

Initially, the a priori input into MAP1, $\mathbf{L^{a1}(U)}$**,** is zero. The output of MAP1, $\mathbf{L^{e1}(U)}$, is interleaved and sent to MAP2 as a prior input $\mathbf{L^{a2}(U)}$. The output of MAP2, $\mathbf{L^{e2}(U)}$**,** is deinterleaved and sent back to MAP1 as a prior input $\mathbf{L^{a1}(U)}$. This “loop” is performed a certain number of iterations. The desired effect is that the extrinsic LLRs increase in absolute value after each iteration. This effectively improves reliability of the decoding, and decreases bit error rate (BER).

It has been showed by exhaustive simulations that, after certain number of iterations, further decoding does not yield any additional BER improvement. In order to reduce processing delay of the decoding process, it is of interest to stop the decoding as soon as this point has been reached. This can be performed by applying a stopping further discussed in section 2.2.5.

After the iterative process has been terminated, the final LLR **L(U)** is computed as a sum of extrinsic information at the output of each MAP decoder, $\mathbf{L^{e1}(U)}$ and $\mathbf{L^{e2}(U)}$, and channel LLR for systematic bits $\mathbf{L^c(U)}$. The final LLR is sent to a threshold device which generates binary (hard) decisions

$\hat{U}$ based on the sign of the LLR, as shown in section 2.2.6.

2.2.1 Channel LLR Computation

Binary outputs of the turbo encoder, denoted **U**, **C** and **C’** in Figure 1, are modulated and send over a noisy channel. Prior to modulation, signal point mapping is performed as follows:

$$U, C, C' = 0 \rightarrow X_{U,C,C'} = -1$$
$$U, C, C' = 1 \rightarrow X_{U,C,C'} = +1$$

If the channel is considered to be fading channel with additive white Gaussian noise, channel outputs corresponding to the systematic bits are:

$$y_U(t) = a_U(t)X_U(t) + n_U(t)$$

where $a_U(t)$ is the fading channel coefficient, and $n_U(t)$ is the noise sample with zero mean and variance No/2. Similarly, the channel outputs corresponding to the parity bits are:

$$y_C(t) = a_C(t)X_C(t) + n_C(t)$$
$$y_{C'}(t) = a_{C'}(t)X_{C'}(t) + n_{C'}(t)$$

If the channel can be assumed to be non-fading, the fading coefficient is $a_U(t) = a_C(t) = a_{C'}(t) = 1$. Alternatively, the fading coefficient, along with noise variance *No* can be approximately determined using channel estimation algorithms.

The channel likelihood for systematic bits $X_U(t)$ is the probability that value $y_U(t)$ has been received, given that $X_U(t)$ has been transmitted, and given that $a_U(t)$ is known. This is computed based on the assumption that $y_U(t) - a_U(t)X_U(t)$ is a Gaussian variable with zero mean and variance *No/2.* The channel likelihood is then computed as follows:

$$\begin{aligned} L^C(U(t)) &= \log \frac{p(y_U(t) |\ X(t) = -1, a_U(t))}{p(y_U(t) |\ X(t) = +1, a_U(t))} \\ &= \log \frac{e^{-(y_U(t) + a_U(t))^2 / No}}{e^{-(y_U(t) - a_U(t))^2 / No}} \\ &= -\frac{4a_U(t)y_U(t)}{No} \\ &= -L^C_U(t)y_U(t) \end{aligned}$$

Similarly, channel likelihoods for parity bits are computed as:

$$L^C(C(t)) = -L^C_C(t)y_C(t),\ L^C(C'(t)) = L^C_{C'}(t)y_{C'}(t)$$

2.2.2 MAP Decoding

The algorithm which is used as a basis for MAP decoding is the Bahl Cocke Jelinek Raviv (BCJR) algorithm first presented in [2], which exploits the trellis structure of a convolutional code. Previous to the discovery of turbo codes, this algorithm has not been used for decoding of convolutional codes due to the availability of a lower complexity Viterbi algorithm (for maximum-likelihood decoding of convolutional codes). The Viterbi algorithm, however, delivers only hard decisions, and not probability distributions (or LLR’s) and can therefore not directly be used in turbo decoding.

The notation for trellis branches used in the subsequent sections is shown in Figure 3. Branch start state is *m’*, and the end state is *m*. The symbol *U(b)* is the input label, i.e., it represents the input into the encoder. The symbol *C(b)* is the output label, or the corresponding output of the encoder which was in state *m’* and received input *U(b).*

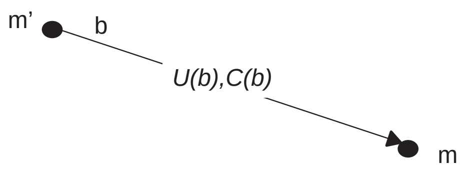

Figure 3. Branch Notation

The BCJR algorithm for MAP decoding of convolutional codes consists of the following steps:

- **Compute branch metric** γ

  This quantity is similar to the local metrics used in Viterbi decoding. The branch metrics represents the logarithm of the probability of branch *b* at time *t*, computed only based on the knowledge of the channel and a priori LLRs of input and output symbols associated with the branch (i.e., not path history through the trellis). For a branch *b* which connects state *m’* at time *(t−1)* and state *m* at time *t*, labeled with input/output pair (*U(b),C(b)*), the branch metric is computed as:

  $$\log \gamma_t(b) = L^C(U(t))U(b) + L^C(C(t))C(b) + L^a(U(t))$$

  where $L^c(U(t))$and $L^c(C(t))$ are systematic and parity channel LLRs, respectively, and $L^a(U(t))$ is a priori LLR.

- **Compute forward state metric** α

  This quantity is similar to accumulated state metrics in Viterbi decoding, and represents log-probability of state *m* at time *t*, given probabilities of states at previous time instances (i.e., knowledge of trellis history). For state *m* at time *t*, the forward state metric is computed as:

  $$\log \alpha_t(m) = \log \sum_{m', \exists(m' \to m)} e^{\log \alpha_{t-1}(m') + \log \gamma_t(b)}$$

  where the summation is performed over all states *m’* at time *t−1* which are connected through a trellis branch to state *m* at time *t*.

- **Compute backward state metrics** β

  This quantity represents the accumulated state metrics, when the trellis is traversed starting from the last stage. It is the log-probability of state *m* at time *t,* given probabilities of states at future time instances (i.e., knowledge of trellis “future”). For state *m* at time *t*, the backward state metric is computed as:

  $$\log \beta_t(m) = \log \sum_{m', \exists(m \to m')} e^{\log \beta_{tk+1}(m') + \log \gamma_{t+1}(b)}$$

  where the summation is performed over all states *m’* at time *t+1* which are connected through a trellis branch to state *m* at time *t*.

- **Compute extrinsic LLR**

  The final output of MAP decoder is obtained by computing the total LLR of bit *U(t)*, given forward and backward state metrics as well as branch metrics for time *t,* and subtracting the systematic channel LLR for bit *U(t), $L^C(U(t))$,* and apriori LLR for bit *U(t), $L^a(U(t))$, both* given at the input to the MAP decoder. The output therefore represents only the “refinement” term, or *extrinsic LLR*:

$$L^e(U(t)) = \log \frac{\sum_{U(b)=1} e^{\log\alpha_{t-1}(m') + \log\beta_t(m) + \log\gamma_t(b)}}{\sum_{U(b)=0} e^{\log\alpha_{t-1}(m') + \log\beta_t(m) + \log\gamma_t(b)}} - L^C(U(t)) - L^a(U(t))$$

where the summations in the numerator and denominator are performed over all branches which are labeled with input labels 1 and 0, respectively.

It can be observed that the most computations involved in MAP decoding are based on the logarithm of a sum of exponentials. Such an expression can be exactly computed, two terms at a time, using the Jacobian logarithm [3]:

$$\log(e^{L_1} + e^{L_2}) = \max(L_1, L_2) + \log(1 + e^{|L_1 - L_2|})$$

i.e., as the maximum exponent and a correction term which is a function of the absolute difference between exponents.

If the correction term is omitted and only max term is used to compute α, β and extrinsic LLR, we obtain the so-called ***max-log-MAP*** approximation.

If the correction term is approximated using a small lookup table, we obtain the ***max*-log-MAP*** approximation. This approximation yields significant BER improvements and is usually preferred over the max-log-MAP solution, in spite of slightly increased computational complexity.

2.2.3 Sliding Window MAP Decoding

One of the drawbacks of the MAP decoding algorithm is its large storage requirements. Forward state metrics α and backward metrics β need to be computed and stored for all states and all stages in the trellis, since they are required for the last step in the algorithm, extrinsic LLR computation. It is possible to combine α or β accumulation with extrinsic information, such that only β or α needs to be stored. This represents $(N+K-1)*2^{(K-1)}$ values which need to be stored, for frame of N information bits and code constraint length K.

In order to reduce memory requirements, large frames can be split into sliding windows, and MAP decoding can be performed on each sliding window independently.

For non-sliding window MAP implementation, α and β are initialized in such a manner that probability 1 is given to $\alpha_0(0)$ and $\beta_{N+K-1}(0)$, since it is known that the initial and the final states of the encoder are zero.

When the frame is split into independent sliding windows, the initial and final states for each window are not known. Therefore, equal probability is given to all α's at the first stage in the window, and all β's at the last stage of the window. In order to achieve reliable decoding, the first segment of α's as well as the last segment of β's should not be used in the final computation of extrinsic information. We call these initial and final segments of the window “header prolog” and “tail prolog”, respectively, and denote them *P*. The extrinsic LLR is only computed over the middle segment of the sliding window, also called “reliability length” and denoted *R*.

In order to obtain extrinsic LLRs for all bits in the frame, the sliding windows should be organized in such a manner that reliability segments of neighbouring windows do not overlap nor form a gap. The header prolog should be overlapped with reliability length of the previous sliding window, and the tail prolog should be overlapped with reliability length of the following sliding window, as shown in Figure 4. Note also that the first window does not require header prolog, since the initial state is known, and for the last window the K−1 tail bits are used instead of the prolog to initialize β's.

With sufficiently large values for P and R, this approach does not result in any BER degradation. A rule of thumb is to use P equal to 3 to 5 times the number of states 2^{K-1}.

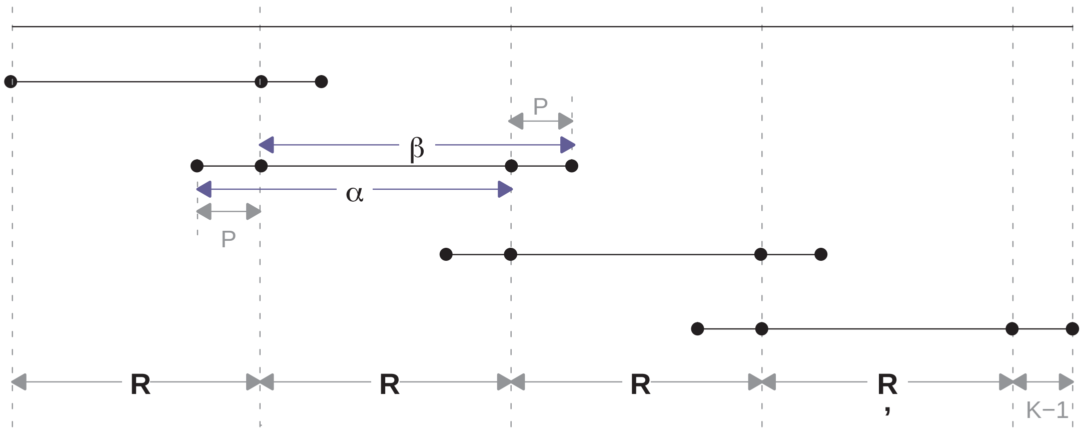

Figure 4. Sliding Window MAP Processing

2.2.4 Interleaving/Deinterleaving

It is necessary to interleave or deinterleave extrinsic LLRs before they can be used as apriori LLRs of the “other” MAP decoder. This function is typically performed using a look-up table. If entry at location “*i*” in the lookup table has value “*ii*”, i.e., *lut[i]=ii*, location “*i*” in the original frame will be copied to location “*ii*” in the interleaved frame. Deinterleaving is performed using the lookup table in the reverse direction, i.e., location “*ii*” in the original frame will be copied to location “*i*” in the deinterleaved frame.

If frames are very long, it may not be feasible to store the interleaver lookup table, in which case the table would need to be generated on the fly.

2.2.5 Stopping Criterion

Research results have confirmed that after a certain number of iterations, there is no benefit in terms of BER of performing additional iterations. The most frequently used number is 6−8 iterations, although the number of useful iterations is proportional to the frame size. A safe approach would be to run the turbo decoder for a fixed (large) number of iterations. This, however, increases processing delay and could waste power.

Recently, algorithms have been devised which can determine automatically if more iterations would yield additional error corrections or not. Such algorithms are called “stopping criteria” and are typically based on some statistics of the extrinsic LLR.

The algorithm used in the TCP implementation is based on the computation of the SNR of extrinsic information, and comparing it against a user-defined threshold.

2.2.6 Hard-Decision Generation

After the last iteration, the final LLR **L(U)** is computed as a sum of systematic channel LLR $\mathbf{L^{C}(U)}$, extrinsic LLR of first MAP, $\mathbf{L^{e1}(U)}$ and deinterleaved extrinsic LLR of second MAP, $\mathbf{L^{e2}(U)}$.

The hard (binary) decision at the output of the turbo decoder is computed based on the sign of the final LLR as follows:

$$\hat{U}(t) = sgn(L(U(t)))$$

3 Relationship Between Turbo Decoding Theory and TCP Implementation

In this section, we establish the relationship between the theory of turbo decoding and the TCP implementation, describing the significance of programmable TCP parameters which affect the TCP algorithm. Those parameters are described in Table 1.

Table 1. Programmable TCP Parameters

Parameter Name	Parameter Description	Applicable modes	Register	Size of parameter (bits)
OPMOD	Operational Mode: − standalone − shared, MAP1 (first iteration) − shared, MAP1 − shared, MAP2	Standalone Shared	TCPIC0	1
RATE	Rate (1/2, 1/3 and 1/4)	Standalone Shared	TCPIC0	2
FL	Frame length	Standalone Shared	TCPIC0	16
R	Reliability Length	Standalone Shared	TCPIC1	7
P	Prolog Length	Standalone Shared	TCPIC2	6
SFL	Sub Frame Length	Shared	TCPIC1	16
NSB	Number of sub-blocks	StandaloneShared	TCPIC2	4
LASTR	Last Sub-Frame Reliability Length	Shared	TCPIC1	7
LASTNSB	Number of sub-blocks for the last sub-frame	Shared	TCPIC2	4
MAXIT	Maximum Number of Iterations	Standalone	TCPIC2	5
SNR	SNR Threshold for the stopping Criterion	Standalone	TCPIC2	8
OUTF	Output Parameters Load Flag	Standalone Shared	TCPIC0	1
INTER	Interleaver Write Flag	Standalone	TCPIC0	1
NWORDSP	Words/XEVT for Systematic/Parities	Standalone Shared	TCPIC3	16
NWORDINTER	Words/XEVT for Interleaver	Standalone	TCPIC3	16
NWORDAP	Words/XEVT for Apriori Write	Shared	TCPIC4	16

Table 1. Programmable TCP Parameters (Continued)

Parameter Name	Parameter Description	Applicable modes	Register	Size of parameter (bits)
NWORDEXT	Words/REVT for Extrinsic Read	Shared	TCPIC4	16
NWORDHD	Words/REVT for Hard Decision Read	Standalone	TCPIC5	16

NOTE: The parameters shown in gray in Table 1 are related to the EDMA operation and do not affect turbo decoding algorithm functionality.

3.1 Code Parameters

TCP supports turbo encoders specified in 3GPP and IS2000 standards. In both cases, 8-state constituent encoders are used. The differences are in code rate, puncturing scheme and handling of tail bits.

The turbo interleaver in the TCP implementation is fully programmable. Depending on the size of the turbo interleaver, the look-up table is either programmed in the TCP interleaver memory, or the interleaving/deinterleaving is performed on the DSP CPU rather than the TCP. The maximum size of the turbo interleaver table which can be input into TCP memory is 5114.

3.1.1 3GPP Codes

For 3GPP, as described in [6], the only turbo code required is a rate 1/3 code with one systematic component **U**, and two coded components, **C** and **C'**, each generated by a recursive constituent encoder with polynomial 15/13. These coded components correspond to **C0** and **C0'** in Figure 1, respectively.

For each frame of data, three tail bits are appended for each top and bottom encoder in order to terminate the trellis in state 0. The tail bits are transmitted along with their coded version, thus resulting in a total of 12 appended bits, in the following order:

$$U(t)C(t)U(t+1)C(t+1)U(t+2)C(t+2)U'(t)C'(t)U'(t+1)C'(t+1)U'(t+2)C'(t+2)$$

where U(t) are tail bits at the input of the top encoder, and C(t) are coded tail bits at the output of the top encoder, and, similarly, U'(t) and C'(t) are tail bits at the input and output of the bottom encoder in Figure 1.

3.1.2 IS2000 Codes

For IS2000, as described in [7], supported codes are rate 1/2, 1/3 and 1/4, all obtained by applying different puncturing scheme to the rate 1/5 turbo encoder shown in Figure 1.

Rate 1/2 turbo code is obtained by using the systematic component **U**, puncturing out all **C1** and **C1'** outputs, corresponding to 17/13 polynomials, and puncturing **C0** or **C0'**, such that the output sequence is {U(t),C0(t), U(t+1),C0'(t+1), U(t+2),C0(t+2),...}. The tail bits are transmitted as follows:

$$U(t)C0(t)U(t+1)C0(t+1)U(t+2)C0(t+2)$$
$$U'(t)C0'(t)U'(t+1)C0'(t+1)U'(t+2)C0'(t+2)$$

Rate 1/3 turbo code is obtained by using the systematic component **U**, puncturing out all **C1** and **C1'** outputs, and always transmitting systematic component **U** and **C0** and **C0'**, such that the output sequence is {U(t),C0(t) ,C0(t) ,U(t+1),C0(t+1), C0'(t+1), U(t+2),C0(t+2), C0'(t+2),...}. The tail bits are transmitted as follows:

$$U(t)U(t)C0(t)U(t+1)U(t+1)C0(t+1)U(t+2)U(t+2)C0(t+2)$$
$$U'(t)U'(t)C0'(t)U'(t+1)U'(t+1)C0'(t+1)U'(t+2)U'(t+2)C0'(t+2)$$

Rate 1/4 encoder is obtained by always transmitting systematic information **U**, the first output of top encoder **C0** and the second output of bottom encoder **C1'**. The remaining two outputs, **C0'** and **C1** are punctured, such that the output sequence is {U(t), C0(t), C0'(t), C1(t), U(t+1), C0(t+1), C1'(t+1), C1(t+1), U(t+2), C0(t+2), C0'(t+2), C1(t+2),...}. Tail bits are transmitted as follows:

$$U(t)U(t)C0(t)C1(t)U(t+1)U(t+1)C0(t+1)C1(t+1)U(t+2)U(t+2)C0(t+2)C1(t+2)$$
$$U'(t)U'(t)C0'(t)C1'(t)U'(t+1)U'(t+1)C0'(t+1)C1'(t+1)U'(t+2)U'(t+2)C0'(t+2)C1'(t+2)$$

3.2 Turbo Decoding Implementation

The role of the TCP in the overall turbo decoder shown in depends on frame size N:

- $N \le 5114$: TCP performs overall turbo decoding. This is called standalone processing mode.
- $N > 5114$: TCP becomes a single MAP decoder. All other operations are performed on DSP CPU. This is called shared processing mode.

Frames smaller than 5114 bits can also be decoded in shared processing mode.

3.2.1 Channel LLR

Systematic and parity channel LLRs, $L^C(U)$ and $L^C(C)$, expected at the input to the TCP are 8-bit signed quantities in (5,3) or 5Q3 fixed point representation. This means that there is a sign bit plus four integer bits, followed by a binary point, followed by three fractional bits (SIIIIFFF), resulting in dynamic range [–32.000, +31.875].. Note that the TCP assumes that the mapping from unsigned to signed binary is $1 \rightarrow -1$ and $0 \rightarrow +1$, and therefore the channel LLR should be computed as shown in section 2.2.1.

3.2.2 MAP Decoding

The MAP decoder implements max*-log-MAP algorithm with a small lookup table.

The computation starts with β computation, with β's for all states and all stages stored to the TCP internal RAM, followed by computation of α and extrinsic LLR. The α values are not stored in memory but are rather used immediately for computation of extrinsic LLR.

β computation is performed traversing the trellis from the end. For the last stage $\beta_{N+K-1}(0)=0.0$, and $\beta_{N+K-1}(m)= -16.000$ for $m \neq 0$. For each state, $\beta_t(m)$ is computed by combining $L^C(X)$, $L^C(Y)$, $L^a(X)$ and previous beta value $\beta_{t+1}(m)$. Normalization is performed at each stage.

The α computation is performed in the identical manner, traversing the trellis from stage 0.

Finally, extrinsic information is obtained combining α, β and apriori inputs. The extrinsic output is a (5,2) fixed point quantity.

3.2.3 Sliding Window Processing

The TCP performs MAP decoding using the sliding window approach described in section 2.2.3. The reliability length R and the prolog length P are programmed for each frame.

The Prolog length P ranges from 24 to 48. For shared processing mode, P must be a multiple of 16. For non-punctured codes, the rule of thumb is to set P to 3 to 5 times the number of states. With 8 states, this number ranges from P=24 to P=40. For punctured codes (i.e., rate 1/2 and 1/4), P should be set to the maximum, i.e., P=48.

Reliability length ranges from 40 to 128, inclusively. The optimal choice for R will be such that the number of sliding windows is minimized, while maximizing parallel processing of sliding windows.

Depending on the frame length, the number of sliding windows which could be processed in parallel is shown in Table 2. A group of sliding windows which are processed in parallel is called a “sub-block”, and is comprised of 1, 2, or 4 sliding windows, as per Table 2.

Table 2. Sliding Window Grouping in Sub-blocks

Frame Length	Number of Sliding Windows per Sub-block, N_SW_SB
40−128	1
129−256	2
257−5114	4

In standalone processing mode, the TCP requires the user to program, for each frame, the prolog length **P**, the reliability length **R**, and the number of sub-blocks **NSB**. Additional value which is not programmed but which is useful in intermediate calculations is the total number of sliding windows per frame, N_SW. The values of **R** and **NSB** are calculated from following equations:

1. N_SW ≥ ceil(**FL**/128)	*Maximum R is 128*
2. N_SW % N_SW_SB = 0	*There is an integer multiple of sub-blocks as per Table 2*
3. FL = (N_SW – 1)*R + R’	*Last sliding window is of length R’, equal or different than R*
4. 40 ≤ R’ ≤ 128, R’ > **P**.	*R’ is such that P for window #(N_SW−1) falls within the frame FL.*
5. NSB = N_SW / N_SW_SB	

In shared processing mode, the frame of length **FL** is split into N_SF sub-frames such that data for one sub-frame fits into the input and output buffers. This requires the length of the sub-frame, **SFL**, to be less than or equal to 5114.

The meaning of the parameter **SFL** required at the input to the TCP is shown in Figure 5.

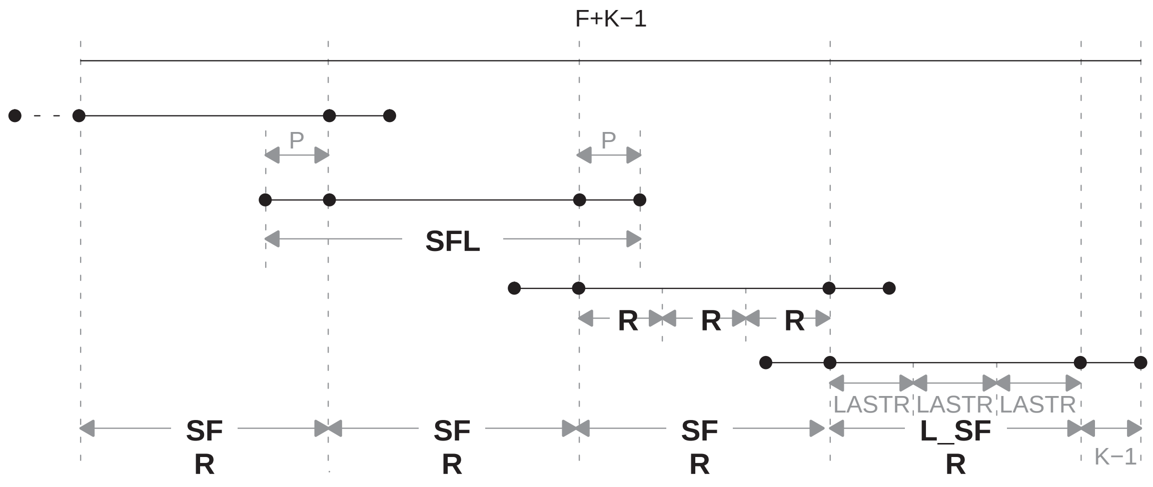

Figure 5. Sliding Window Processing in Shared-Processing Mode

Inside each sub-frame, the processing is still split into sliding windows of size **R** defined earlier, and Table 2 still applies for each sub-frame. It can be seen that the value **SFL** actually includes the header prolog of the first sliding window inside the sub-frame, and the tail prolog of the last sliding window inside the sub-frame.

It is useful to define a quantity SFR = **SFL** – 2***P** which defines the length over which the extrinsic information is computed. Then, **SFL** is computed from the following conditions:

6. N_SF = ceil(**FL**/5114)	*Data must fit in TCP input buffers*
7. SFR = **SFL** – 2***P**	*Portion over which extrinsic information is computed*
8. FL = (N_SF –1)*SFR + L_SFR	
9. r=1/2,1.3: SFR % 8, P%8 = 0	*Due to EDMA indexing*
10. r=1/4: SFR % 16, P%16 = 0	*Due to EDMA indexing*

Values of **NSB** and **R** are computed as for the standalone mode, using SFR instead of **FL** in line (3) above.

The length of the last sub-frame, L_SFR, could be smaller than the remaining subframes. The value of L_SFR is not input into the TCP. However, parameters of sliding window processing within the last sub-frame may be redefined as a consequence of a different sub-frame length. These parameters are input into the TCP as **LASTR** and **LASTNSB**, for the last sub-frame reliability and the number of sub-blocks, respectively. **LASTR** and **LASTNSB** are computed in the same manner as **R** and **NSB** for standalone mode, using L_SFR instead of **FL** in equation (3) above.

3.2.4 Interleaving/Deinterleaving

Interleaving or deinterleaving is performed in the TCP in the standalone processing mode only, for frames up to 5114 bits. The look up table is generated by the DSP CPU and transferred to the TCP interleaver memory as a part of the TCP initialization process, as 16-bits per table entry.

This transfer is optional and determined by the TCP from the value of **INTER** (interleaver write flag). If the frame to be decoded uses the same turbo interleaver as the frame previously decoded, the transfer of interleaver is table can be omitted.

In shared processing mode, the interleaving and deinterleaving is done by the CPU.

3.2.5 Stopping Criterion

The stopping criterion implemented in the TCP is used in standalone processing only. It is based on measurement of extrinsic SNR, comparison against a user-defined threshold, and stopping the iterative processing if the threshold has been reached or exceeded. The SNR of extrinsic information is computed over the entire frame. The user defined threshold which the computed SNR value has to exceed in order for the iterations to stop, is input into the TCP for each frame and can range from 1 to 100.

Setting **SNR** to zero disables the stopping criterion, so that the maximum number of iterations **MAXIT** specified for that particular frame will always be executed.

3.2.6 Hard-Decision Decoding

Hard decision decoding is performed as described in section 2.2.6.

For standalone processing, the hard decision decoding is performed in the TCP. For shared processing, it is part of DSP CPU processing.

4 TCP Programming Procedure

This section outlines steps required to decode a single frame of data using the TCP. For possible approaches to decoding of multiple frames of the same or different user channels, see section 6.

4.1 Standalone Processing

In standalone processing, the TCP implements functionality of the entire turbo decoder. The programming procedure for decoding one frame of data is described in this section.

4.1.1 Initialize Input Buffers

Input buffers consist of normalized systematic and parity LLR’s, computed from channel soft decisions, as well as the interleaver look-up table. The organization of the systematic and parities expected at the input of TCP is given in []. For a frame with **FL** information bits, the total size of systematic and parity LLR array is **FL***RATE**. Each LLR is an 8-bit signed value. The number of entries in the interleaver look-up table is equal to the frame length **FL**, each entry being a 16-bit unsigned value. The DSP memory addresses of the beginning of the systematic and parities, and interleaver buffers will be referred to as &s_p[0] and &int[0], respectively. All data is aligned on a doubleword (64-bit) boundary.

4.1.2 Allocate Memory for Output Buffers

The output buffer is allocated for hard decisions, which are packed into 64-bit words. For a frame length **FL**, the size of allocated buffer should be ceil(**FL**/64)*8 bytes, aligned on a 64-bit boundary. In addition, if the output parameter read flag is set (**OUTF**=1), one 64-bit word should be allocated for it. The DSP memory addresses of the beginning of the allocated buffers for TCP decisions and output parameters will be referred to as &hd[0] and &out_p[0], respectively.

4.1.3 Prepare Input Configuration

The input configuration consists of three sections: (1) TCP Configuration (2) EDMA Interface Configuration and (3) Tail bits, and is programmed through registers TCPIC0−TCPIC11. The input configuration is first prepared in the DSP memory (internal or external). It is transferred to the TCP via EDMA once the TCP is started. The DSP memory address of the beginning of the prepared configuration is denoted &inconf[0].

4.1.4 Prepare EDMA Links

The TCP generates a series of receive events (TCPREVT) when it is ready to send data to the EDMA, and a series of transmit events (TCPXEVT) when it is ready to receive data from the EDMA. Each event has an EDMA channel associated with it, and for each event certain number of EDMA links is required.

For standalone processing, the TCP requires input configuration, systematic and parities, and, optionally, interleaver. The TCP generates hard decisions and, optionally, output parameters. REVTs and XEVTs are generated for each of the above data sets, and therefore EDMA linking is used. EDMA links are summarized in Table 3. For each EDMA transfer, programmable parameters are: (1) transfer options, (2) source address, (3) destination address, (4) frame and element count, (5) frame and element index, (6) reload count and link address. These parameters are described in detail in [5].

The third row of the table represents the address in the paRAM. Link 0 of each TCPXEVT and TCPREVT have to be programmed at fixed locations in the paRAM, denoted as ADDR_TCPXEVT and ADDR_TCPREVT, respectively. Other linkes could be programmed anywhere in the paRAM. There additional locations in the paRAM are denoted RELOAD1, RELOAD2 and RELOAD3.

The LINK entry in each parameter set represents the paRAM address of the next linked transfer. LINK=NULL indicates that the next transfer is the NULL transfer used for termination. Element count ELECNT and frame count FRMCNT are also computed.

Table 3. EDMA Links for Standalone Processing

TCPXEVT Links			TCPREVT Links	
Link 0	**Link 1**	**Link 2 (optional)**	**Link 0**	**Link 1 (optional)**
paRAM address = ADDR_TCPXEVT	paRAM address = RELOAD1	paRAM address = RELOAD2	paRAM address = ADDR_TCPREVT	paRAM address = RELOAD3
OPT: SUM=DUM=INC	OPT: SUM=INC, DUM=FIXED	OPT: SUM=INC, DUM=FIXED	OPT: SUM=FIXED, DUM=INC	OPT: SUM=DUM=INC TCINT=1, TCC = TCPREVT
SRC= &inconf[0]	SRC= &s_p[0]	SRC= &int[0]	SRC= TCPHD	SRC= TCPOUT
FRMCNT = 0 ELECNT = 12	FRMCNT ELECNT	FRMCNT ELECNT	FRMCNT ELECNT	FRMCNT = 0 ELECNT = 2
DST= TCPIC0	DST= TCPSP	DST= TCPINTER	DST= &hd[0]	DST= &out_p[0]
FRMIDX = N/A ELEIDX = N/A	FRMIDX = N/A ELEIDX = N/A	FRMIDX = N/A ELEIDX = N/A	FRMIDX = N/A ELEIDX = N/A	FRMIDX = N/A ELEIDX = N/A
ELERLD = N/A LINK = RELOAD1	ELERLD = N/A LINK = RELOAD2 (INTER=1) = NULL (INTER=0)	ELERLD = N/A LINK= NULL	ELERLD = N/A LINK = RELOAD3 (OUTF=1) = NULL (OUTF=0)	ELERLD = N/A LINK = NULL

4.1.5 Start EDMA and Enable Interrupts

The EDMA channels corresponding to TCP's TCPREVT and TCPXEVT are enabled in EDMA Event Enable Register (EER), and these channels are also allowed to generate CPU interrupts by setting appropriate bits in Channel Interrupt Enable Register (CIER). The EDMA control registers are described in detail in [5].

4.1.6 Start TCP

The CPU writes a START command into the TCP's execution word register (TCPEXE). This causes the TCP to generate the first TCPXEVT expecting input configuration. This in turn will trigger the EDMA transfer which is programmed into the Event paRAM location corresponding to TCPXEVT.

4.1.7 Service EDMA Interrupt from TCP Channel at the End of Decoding

The EDMA link associated with the last TCPREVT should be configured as to generate a CPU interrupt. In the CPU interrupt service routine, the output decision buffer for the completed frame can be processed. For example, the ISR could compute the cyclic-redundancy check (CRC) or schedule further processing , and initiate decoding of the next frame.

4.2 Shared Processing

In shared processing mode, the TCP implements functionality of a single MAP decoder, i.e., MAP1 or MAP2 from . The DSP is responsible for interleaving/deinterleaving of extrinsic information, the stopping criterion, and the computation of hard decisions.

In this section, we describe the TCP setup procedure for MAP1 or MAP2.

4.2.1 Initialize Input Buffers

Input buffers consist of normalized systematic and parity LLR’s, computed from channel soft decisions, as well as apriori LLRs. The organization of systematic and parities expected at the input of the TCP is described in []. For a frame with **FL** information bits, the total size of systematic and parity LLR array for rate 1/2 and 1/3 is 2***FL**, and for rate 1/4 is ceil(2.5***FL**). The number of apriori LLR is **FL**. Each LLR is an 8-bit signed value, computed as shown in section 2.2.1 The DSP memory addresses of the beginning of the systematic and buffer will be referred to as &s_p[0]. All data is aligned on a doubleword (64-bit) boundary.

4.2.2 Allocate Memory for Output Buffers

The output buffer is for extrinsic LLR, which is used as the apriori input for the next MAP decoder. For a frame length FL, the allocated buffer for extrinsic LLR should be **FL** bytes, aligned on a 64-bit boundary. The size of the allocated buffer should be 8*ceil(**FL**/8). The DSP memory addresses of the beginning of the allocated buffer extrinsic outputs will be referred to as &ext[0].

4.2.3 Prepare TCP Input Configuration Word

The input configuration consists of three sections: (1) TCP Configuration (2) EDMA Interface Configuration and (3) Tail bits, and is programmed through registers TCPIC0−TCPIC11. The input configuration is first prepared in the DSP memory (internal or external). It is transferred to the TCP via EDMA once the TCP is started. The DSP memory address of the beginning of the prepared configuration is denoted &inconf[0].

4.2.4 Prepare EDMA Links

In general, the interface between the TCP and EDMA is identical to the interface described for the standalone mode at the beginning of section 4.1.4 and the description of the common setup for the EDMA will not be discussed here separately.

An EDMA feature used in shared processing mode is the channel chaining capability. It is used instead of channel linking in order to transfer apriori LLRs into the TCP. Another feature used in shared processing mode is indexed address update mode. It is used for transfer of apriori LLRs and systematic and parity LLRs.

A summary of EDMA transfer parameters for shared processing mode is shown in Table 4, and details are given in [4].

Table 4. EDMA Links for Shared Processing (per MAP)

TCPXEVT Links						TCPREVT Links	
Link 0		Link 1		(CHAIN on CH_AP)		Link 0	
paRAM address = ADDR_TCPXEVT		paRAM address = RELOAD1		paRAM address = RELOAD2		paRAM address = ADDR_TCPREVT	
OPT: SUM=DUM=INC		OPT SUM=INDX, DUM=FIXED, ATCC=CH_AP		OPT: SUM= INDX, DUM=FIXEDT		OPT: SUM=FIXED, DUM=INC TCINT=1, TCC = TCPREVT	
SRC= &inconf[0]		SRC= &s_p[0]		SRC= &ap[0]		SRC=TCPEXT	
FRMCNT = 0	ELECNT = 12	FRMCNT	ELECNT	FRMCNT	ELECNT	FRMCNT	ELECNT
DST=TCPIC0		DST=TCPSP		DST=TCPAP		DST= &ext[0]	
FRMIDX = N/A	ELEIDX = N/A	FRMIDX	ELEIDX = 4	FRMIDX	ELEIDX = 4	FRMIDX = N/A	ELEIDX = N/A
ELERLD = N/A	LINK = RELOAD1	ELERLD = N/A	LINK = NULL	ELERLD = N/A	LINK = NULL	ELERLD = N/A	LINK = NULL

4.2.5 Start EDMA and Enable Interrupts

This setup is performed in the same manner as for the standalone mode described in section 4.1.5. One additional operation required is to enable transfer chaining for CH_AP by setting the appropriate bit in the Channel Chain Enable Register (CCER).

4.2.6 Start TCP

This setup is the same as for standalone mode, described in section 4.1.6.

4.2.7 Service EDMA Interrupt from TCP Channel at the End of Decoding

The EDMA link associated with the last TCPREVT should be configured as to generate a CPU interrupt. In the CPU interrupt service routine, the output decision buffer for the completed frame can be processed. For example, the ISR could perform or schedule interleaving/deinterleaving of the apriori buffer, decide if additional iterations are required, and initiate next MAP decoding, or compute hard decisions.

5 Programming Examples

In this section we show how to program the TCP input configuration parameters as well as EDMA links, for a single frame of data with typical 3G wireless decoding parameters.

For each example, we will discuss how to determine the TCP/EDMA configuration parameters. The EDMA link configuration for TCPXEVT Link 0 (write to TCP input configuration) and TCPREVT Link 1 (read from TCP output parameters) for standalone mode are as shown in Table 3 and for shared processing in Table 4. They will not be repeated in the examples, but it is understood that they need to be programmed.

5.1 Standalone Mode Examples

5.1.1 IS2000, 378-Bit Frame, Rate 1/4

This example is for an IS2000 378 bit-frame, encoded using rate 1/4 code. The TCP can be used in standalone mode.

The Prolog length is set to the maximum, i.e., **P**=48, since the code is punctured. Stopping criterion will be enabled with threshold **SNR**=50, and up to **MAXIT**=8 iterations will be executed. Since the stopping criterion is enabled, we may be interested in the number of iterations executed so the output parameters will be read, i.e., **OUTF**=1.

Reliability length **R** and number of sub-blocks **NSB** are computed as shown in section 3.2.3:

1. N_SW ≥ ceil(378/128) = 3
2. N_SW % 4 = 0
3. FL = (N_SW – 1)*R + R’
4. 40 ≤ R’ ≤ 128, R’ > **P**.
5. NSB = N_SW / 4

From (1) and (2), N_SW=4. Equation (3) is then satisfied with **R**=95 (R’=93). From (5), **NSB**=1.

Assuming all transfers are performed as a single EDMA frame, the EDMA interface is configured as follows:

- NWORDINTER = ceil(378/2)=189
- NWORDSP = ceil((378*4)/4)=378
- NWORDHD = ceil(378/32)=12

For systematic and parity transfer ELECNT=2*ceil(**NWORDSP**/2)=378 and FRMCNT=0, for interleaver transfer ELECNT=2*ceil(**NWORDINTER**/2)=190 and for hard decision transfer ELECNT=2*ceil(**NWORDHD**/2**)**=12 and FRMCNT=0. Link 0 for TCPREVT is linked to Link 1 since output parameters are transferred.

These parameters are summarized in Table 5.

Table 5. TCP/EDMA Configuration for 378-Bit Frame, Rate 1/4 (IS2000)

TCP Configuration					
OPMOD = standalone (000b)			RATE = 1/4		
FL = 378			R = 95		P = 48
NSB = 1			MAXIT= 8		SNR = 50
SFL = N/A			LASTR = N/A		LASTNSB = N/A
EDMA Interface Configuration					
OUTF= 1			INTER = 1		
NWORDSP= 378					
NWORDINTER = 189			NWORDHD = 12		
NWORDHD = N/A			NWORDEXT = N/A		
Tail Bits					
X(t)	X(t+1)	X(t+2)	Y0(t)	Y0(t+1)	Y0(t+2)
Y1(t)	Y1(t+1)	Y1(t+2)	X’(t)	X’(t+1)	X’(t+2)
Y0’(t)	Y0’(t+1)	Y0’(t+2)	Y1’(t)	Y1’(t+1)	Y1’(t+2)

TCPXEVT Links				**TCPREVT links**	
Link 1		**Link 2**		**Link 0**	
OPT = see Table 3		OPT= see Table 3		OPT= see Table 3	
SRC= &s_p[0]		SRC= &int[0]		SRC= TCPHD	
FRMCNT= 0	ELECNT= 378	FRMCNT= 0	ELECNT= 189	FRMCNT= 0	ELECNT= 12
DST= TCPSP		DST= TCPINTER		DST= &hd[0]	
FRMIDX= N/A	ELEIDX = N/A	FRMIDX= N/A	ELEIDX = N/A	FRMIDX = N/A	ELEIDX = N/A
ELERLD= N/A	LINK= 2	ELERLD= N/A	LINK= NULL	ELERLD= N/A	LINK= 1

5.1.2 3GPP, 3840-Bit Frame, Rate 1/3

This example is for a 3GPP 3480 bit-frame, encoded using rate 1/3 code. The TCP can be used in standalone mode.

The prolog length can be set to a minimum, i.e., **P**=24, since the code is not punctured. Stopping criterion will be enabled with threshold **SNR**=50, and up to **MAXIT**=16 iterations will be executed. Since the stopping criterion is enabled, we may be interested in the number of iterations executed so the output parameters will be read, i.e., **OUTF**=1.

Reliability length **R** and number of sub-blocks **NSB** are computed as shown in section 3.2.3:

1. N_SW ≥ ceil(3840/128) = 30
2. N_SW % 4 = 0
3. FL = (N_SW – 1)*R + R’
4. 40 ≤ R’ ≤ 128, R’ > **P**.
5. NSB = N_SW / 4

From (1) and (2), N_SW=32. Equation (3) is then satisfied with **R**=120 (R’=120). From (5), **NSB**=8.

Assuming all transfers are performed as a single EDMA frame, the EDMA interface is configured as follows:

- NWORDINTER = ceil(3840/2)=1920
- NWORDSP = ceil(ceil((3840*3)/4)=2880
- NWORDHD = ceil(3840/32)=120

For systematic and parity transfer, ELECNT=2*ceil(**NWORDSP**/2)=2880 and FRMCNT=0, for interleaver transfer, ELECNT=2*ceil(**NWORDINTER**/2)=1920 and for hard decision transfer ELECNT=2*ceil(**NWORDHD**/2**)**=120 and FRMCNT=0. Link 0 for TCPREVT is linked to Link 1 since output parameters are transferred.

These parameters are summarized in Table 6.

Table 6. TCP/EDMA Configuration for 3840-Bit Frame, Rate 1/3 (3GPP)

TCP Configuration							
OPMOD = standalone (000b)			RATE = 1/3				
FL = 3840			R = 120			P = 24	
NSB = 8			MAXIT = 16			SNR = 50	
SFL = N/A			LASTR = N/A			LASTNSB = N/A	
EDMA Interface Configuration							
OUTF= 1			INTER = 1				
NWORDSP= 2880							
NWORDINTER = 1920			NWORDHD = 120				
NWORDHD = N/A			NWORDEXT = N/A				
Tail Bits							
X(t)	X(t+1)	X(t+2)	0	Y0(t)	Y0(t+1)	Y0(t+2)	0
0	0	0	0	X’(t)	X’(t+1)	X’(t+2)	0
Y0’(t)	Y0’(t+1)	Y0’(t+2)	0	0	0	0	0
TCPXEVT Links						**TCPREVT links**	
Link 1			**Link 2**			**Link 0**	
OPT = see Table 3			OPT= see Table 3			OPT= see Table 3	
SRC= &s_p[0]			SRC= &int[0]			SRC= TCPHD	
FRMCNT= 0		ELECNT= 2880	FRMCNT= 0		ELECNT = 1920	FRMCNT = 0	ELECNT = 120
DST= TCPSP			DST= TCPINTER			DST= &hd[0]	
FRMIDX= N/A		ELEIDX = N/A	FRMIDX= N/A		ELEIDX = N/A	FRMIDX = N/A	ELEIDX = N/A
ELERLD= N/A		LINK= 2	ELERLD = N/A		LINK = NULL	ELERLD = N/A	LINK = 1

5.1.3 3GPP, 5114 Frame, Rate 1/2

This example is for a 3GPP 5114 bit-frame, encoded using rate 1/2 code. This is the largest frame supported in the standalone mode.

The prolog length is set to a maximum, i.e., **P**=48, since the code is punctured. Stopping criterion will be enabled with threshold **SNR**=50, and up to **MAXIT**=16 iterations will be executed. Since the stopping criterion is enabled, we may be interested in the number of iterations executed so the output parameters will be read, i.e., **OUTF**=1.

Reliability length **R** and number of sub-blocks **NSB** are computed as shown in section 3.2.3:

1. N_SW ≥ ceil(5114/128) = 40
2. N_SW % 4 = 0
3. FL = (N_SW – 1)*R + R’
4. 40 ≤ R’ ≤ 128, R’ > **P**.
5. NSB = N_SW / 4

From (1) and (2), N_SW=40. Equation (3) is then satisfied with **R=128** (R’=122). From (5), **NSB=10**.

Assuming all transfers are performed as a single EDMA frame, the EDMA interface is configured as follows:

- NWORDINTER = ceil(5114/2)=2557
- NWORDSP = ceil((5114*2)/4)=2557
- NWORDHD = ceil(5114/32)=160

For systematic and parity transfer, ELECNT=2*ceil(**NWORDSP**/2)=2558 and FRMCNT=0, for interleaver transfer, ELECNT=2*ceil(**NWORDINTER**/2)=2558 and for hard decision transfer ELECNT=2*ceil(**NWORDHD**/2**)**=160 and FRMCNT=0. Link 0 for TCPREVT is linked to Link 1 since output parameters are transferred.

These parameters are summarized in Table 7.

Table 7. TCP/EDMA Configuration for 5114-Bit Frame, Rate 1/2 (3GPP)

TCP Configuration							
OPMOD = standalone (000b)			RATE = 1/2				
FL = 5114			R = 128			P = 48	
NSB = 10			MAXIT = 16			SNR = 50	
SFL = N/A			LASTR = N/A			LASTNSB = N/A	
EDMA Interface Configuration							
OUTF= 1			INTER = 1				
NWORDSP= 2557							
NWORDINTER = 2557			NWORDHD = 160				
NWORDHD = N/A			NWORDEXT = N/A				
Tail Bits							
X(t)	X(t+1)	X(t+2)	0	Y0(t)	Y0(t+1)	Y0(t+2)	0
0	0	0	0	X’(t)	X’(t+1)	X’(t+2)	0
Y0’(t)	Y0’(t+1)	Y0’(t+2)	0	0	0	0	0

TCPXEVT Links				**TCPREVT links**	
Link 1		**Link 2**		**Link 0**	
OPT = see Table 3		OPT= see Table 3		OPT= see Table 3	
SRC= &s_p[0]		SRC= &int[0]		SRC= TCPHD	
FRMCNT= 0	ELECNT= 2558	FRMCNT = 0	ELECNT = 2558	FRMCNT = 0	ELECNT = 160
DST= TCPSP		DST= TCPINTER		DST= &hd[0]	
FRMIDX= N/A	ELEIDX = N/A	FRMIDX = N/A	ELEIDX = N/A	FRMIDX = N/A	ELEIDX = N/A
ELERLD= N/A	LINK= 2	ELERLD = N/A	LINK = NULL	ELERLD = N/A	LINK = 1

5.2 Shared Processing Examples

5.2.1 IS2000, 6138-Bit Frame, Rate 1/2

This example is for an IS2000 6138 bit-frame, encoded using rate 1/2 code. Since **FL** > 5114, shared processing mode must be used. The TCP programming parameters will be shown for the first MAP1 and the first MAP2 (i.e., first iteration).

The prolog length is set to a maximum, i.e., **P**=48, since the code is punctured.

SFL is computed as shown in equations 6−9 in section 3.2.3:

6. N_SF = ceil(6138/5114)	*Data must fit in TCP input buffers*
7. SFR = **SFL** – 2*48	*Portion over which extrinsic information is computed*
8. FL = (N_SF –1)*SFR + L_SFR	
9. SFR % 4 = 0	*Due to EDMA indexing features*

From (6) , N_SF=2. Then, (8) and (9) are satisfied with SFR=3072, L_SFR=3066. From (7), **SFL**=3168.

Reliability length **R** and number of sub-blocks **NSB** are computed as shown in section 3.2.3 using SFR=3072 instead of **FL**:

1. $N\_SW \geq ceil(3072/128) = 24$
2. $N\_SW \% 4 = 0$
3. SFR = (N_SW – 1)***R** + R’
4. 40 ≤ R’ ≤ 128, R’ > **P**.
5. NSB = N_SW / 4

From (1) and (2), N_SW=24. Equation (3) is then satisfied with **R**=128 (R’=128). From (5), **NSB**=6.

For the last subframe, Reliability length **LASTR** and number of sub-blocks **LASTNSB** are computed as shown in section 3.2.3 using L_SFR=3066 instead of **FL**:

1. $N\_SW \geq ceil(3066/128) = 24$
2. $N\_SW \% 4 = 0$
3. L_SFR = (N_SW – 1)***LASTR** + R’
4. 40 ≤ R’ ≤ 128, R’ > **P**.
5. LASTNSB = N_SW / 4

From (1) and (2), N_SW=24. Equation (3) is then satisfied with **LASTR**=128 (R’=128). From (5), **LASTNSB**=6.

The EDMA interface is configured as follows:

- NWORDSP = (SFL*2)/4 = (3168*2)/4 = 1584
- NWORDAP =SFL/4 =3186/4 = 792
- NWORDEXT = SFR/4 = 3072/4 = 768

The EDMA transfer for systematic and parity LLRs is configured with ELECNT = 2*ceil(**NWORDSP**/2) = 1584, FRMCNT = N_SF−1=1 and FRMIDX=SFR*2=6144.

The EDMA transfer for apriori LLR (used in all modes except MAP 1 for first iteration) is configured with ELECNT = 2*ceil(**NWORDAP**/2) = 792, FRMCNT = N_SF−1=1 and FRMIDX=SFR=3072.

The EDMA transfer for extrinsic LLR is configured with with ELECNT = 2*ceil(**NWORDEXT**/2) = 768 and FRMCNT = N_SF−1=1 .

The parameters for MAP 1 (first iteration) and MAP 2 (any iteration) are summarized in Table 8 and Table 9.

Table 8. TCP/EDMA Configuration for 6138-Bit Frame, Rate 1/2 (IS2000) – MAP 1 (first iteration)

TCP Configuration							
OPMOD = shared, MAP 1 (no context)			RATE= 1/2				
FL = 6138			R = 128			P = 48	
NSB = 6			MAXIT = N/A			SNR = N/A	
SFL = 3168			LASTR = 128			LASTNSB = 6	
EDMA Interface Configuration							
OUTF= 0			INTER = 0				
NWORDSP= 1584							
NWORDINTER = N/A			NWORDHD = N/A				
NWORDHD = N/A			NWORDEXT = 768				
Tail Bits							
X(t)	X(t+1)	X(t+2)	0	Y0(t)	Y0(t+1)	Y0(t+2)	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

TCPXEVT Links			**TCPREVT links**	
Link 1		**Link 2 − Not Used**	**Link 0**	
OPT: see Table 4			OPT: see Table 4	
SRC= &s_p1[0 – 2*48]			SRC= TCPEXT	
FRMCNT= 1	ELECNT= 1584		FRMCNT = 1	ELECNT = 768
DST= TCPSP			DST= &ext[0]	
FRMIDX= 6144	ELEIDX = 4		FRMIDX = N/A	ELEIDX = N/A
ELERLD= N/A	LINK= NULL		ELERLD = N/A	LINK = NULL

Table 9. TCP/EDMA Configuration for 6138-Bit Frame, Rate 1/2 (IS2000) – MAP 2

TCP Configuration							
OPMOD = shared, MAP 2			RATE = 1/2				
F = 6138			R = 128			P = 48	
NSB = 6			MAXIT = N/A			SNR = N/A	
SFL = 3168			LASTR = 128			LASTNSB = 6	
EDMA Interface Configuration							
OUTF= 0			INTER = 0				
NWORDSP= 1584							
NWORDINTER = N/A			NWORDHD = N/A				
NWORDHD = 792			NWORDEXT = 768				
Tail Bits							
X’(t)	X’(t+1)	X’(t+2)	0	Y0’(t)	Y0’(t+1)	Y0’(t+2)	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
TCPXEVT Links						**TCPREVT links**	
Link 1			**Link 2 − Not Used**			**Link 0**	
OPT: see Table 4			OPT: see Table 4			OPT: see Table 4	
SRC= &s_p2[0 – 2*48]			SRC= &ap [0 − 48]			SRC= TCPEXT	
FRMCNT= 2−1	ELECNT= 1584		FRMCNT = 2−1		ELECNT = 792	FRMCNT = 1	ELECNT = 768
DST= TCPSP			DST= TCPAP			DST= &ext[0]	
FRMIDX= 6144	ELEIDX = 4		FRMIDX = 3072		ELEIDX = 4	FRMIDX = N/A	ELEIDX = N/A
ELERLD= N/A	LINK= 2		ELERLD =		LINK = NULL	ELERLD = N/A	LINK = NULL

6 Multichannel Operation Considerations

The coprocessor will typically be used in an operating environment where a series of frames is to be decoded in a most efficient manner. The efficiency could be with respect to one or more of the following parameters:

- **EDMA paRAM space:** Each frame of data requires a certain number of links in the paRAM, and due to the limited size of the paRAM, it may not be feasible to pre-program EDMA links for all frames to be decoded.
- **CPU interrupt rate:** The CPU intervention may be required to initialize input buffers for new frames to be decoded, process decoded frames, and program new EDMA links.
- **Percentage of coprocessor capabilities required:** If the processing power of the coprocessor is pushed to the maximum, then frame decoding should be scheduled in such a manner as to keep the coprocessor constantly active, i.e., not let it wait for new input data to be transferred in, or decoded data to be transferred out.

In this section, we discuss several approaches to scheduling decoding of a series of frames. Each method optimizes one of the above mentioned parameters.

6.1 Method 1: paRAM-Efficient

This method is the most simple to program and requires the least number of links in the EDMA paRAM.

The paRAM usage and EDMA linking is shown in Figure 6 and Figure 7 for standalone and shared processing mode, respectively.

Assuming that the coprocessor is initially idle (i.e., in “RESET” state), the suggested procedure is as follows:

1. CPU programs Link 0 for TCPREVT and Link 1 for TCPXEVT into the appropriate Event location in paRAM. The remaining links are programmed anywhere in the paRAM space. Note that , in shared processing mode, EDMA link responsible for Apriori links (TCPXEVT Link 2) should be in the first 64 links in the paRAM (due to chaining requirement). The programmed EDMA transfers are inline with those outlined in examples in the previous section, i.e.,:
   − The last link for TCPXEVT is linked to a NULL transfer, and does not generate a CPU interrupt.
   − The last link for TCPREVT is also linked to a NULL transfer, and it generates a CPU interrupt with TCC which correspond to TCPREVT.
2. The CPU configures EDMA interrupt generation such that TCPREVT is enabled, and EDMA interrupts are enabled.
3. The CPU sends “START” command to the TCP and continues any non-interfering processing.
4. When CPU receives EDMA interrupt, with TCC=TCPREVT, the CPU performs necessary input/output buffer management and repeats steps (1)−(3) for the next frame.

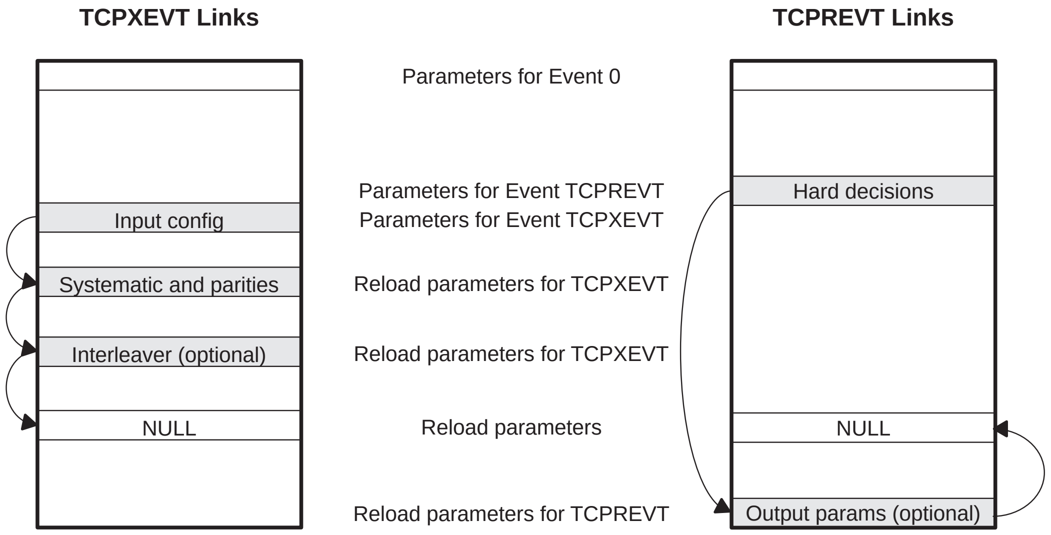

Figure 6. Method 1: paRAM Entries for Standalone

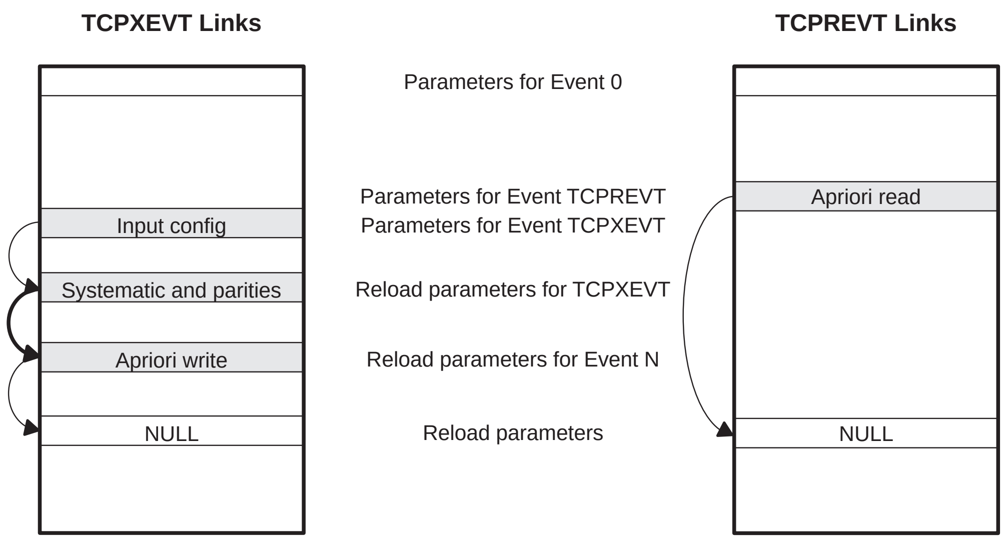

Figure 7. Method 1: paRAM Entries for Shared Processing (With Previous Context)

6.2 Method 2: Continuous Decoding

The main problem with the approach outlined in section 6.1 is that the coprocessor is kept waiting for input data, i.e., decoding of a new frame does not start as soon as the coprocessor is ready for it.

This problem could be remedied by keeping EDMA links for two frames in the paRAM. The CPU is interrupted once frame #n has been decoded, in order to program links for frame #(n+2). Meanwhile, the coprocessor is processing frame #(n+1). This assumes that the TCP processing delay for frame #(n+1) is sufficiently large such that the CPU has enough time to respond to interrupt and write links for frame #(n+2) into the paRAM.

This concept is illustrated in Figure 8 and Figure 9 for standalone and shared processing, respectively.

The procedure is as follows:

1. The CPU programs all links for the first two frames. Note that the first TCPREVT and first TCPXEVT link for frame #1 is written into Event parameters, and not Reload parameters.
2. The CPU configures EDMA interrupt generation such that TCPREVT is enabled, and EDMA interrupts are enabled.
3. The CPU sends “START” command to the TCP and continues any non-interfering processing
4. Once the transfer associated with link called “Output Params (optional) # 1, 3, 5,...” is completed, the CPU interrupt is generated. The CPU overwrites links associated with channel #1 with those associated with channel #3. Note that that the first TCPREVT and first TCPXEVT link for frame #3 are written into Reload parameters, since the Event parameter space is used by the link currently in progress.
5. Once the transfer associated with link called “Output Params (optional) # 2, 4, 6,...” is completed, the CPU interrupt is generated. The CPU overwrites links associated with channel #2 with those associated with channel #4.

6. Steps (4) and (5) are repeated as long as there are frames to be processed. If the CPU gets interrupted and there are no additional frames to be processed, the last previously programmed link for both TCPREVT and TCPXEVT should be relinked to the NULL parameter set to terminate the transfer.

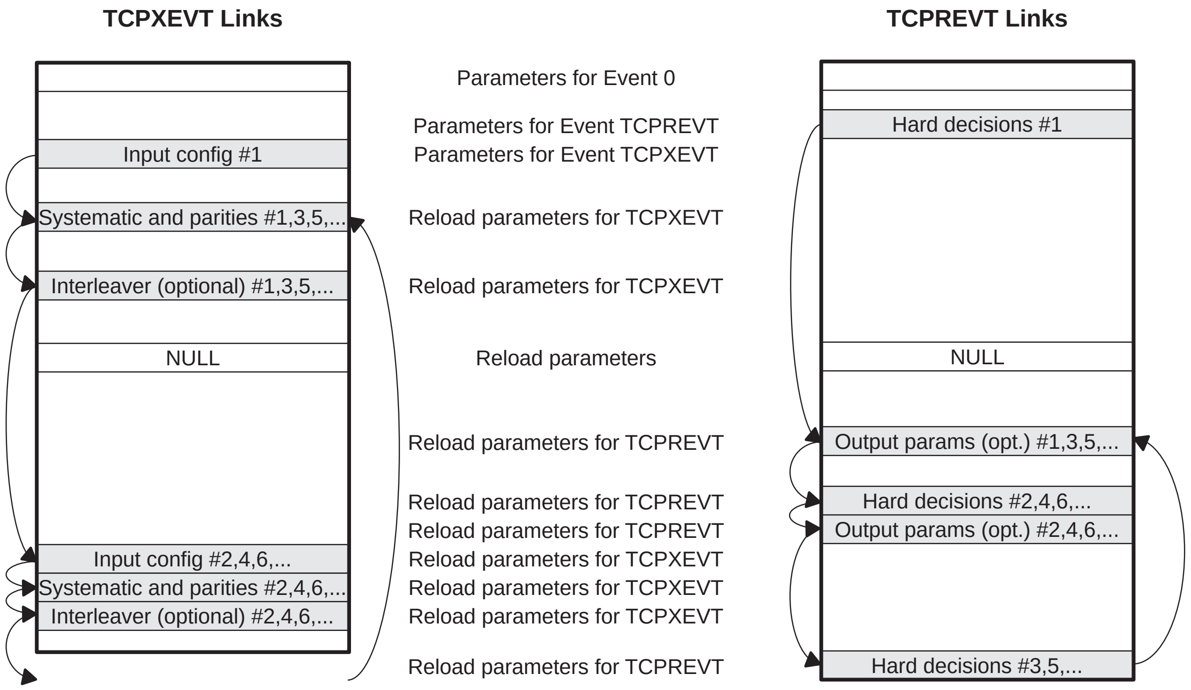
TCPXEVT Links
TCPREVT Links
Parameters for Event 0
Parameters for Event TCPREVT
Parameters for Event TCPXEVT
Reload parameters for TCPXEVT
Reload parameters for TCPXEVT
Reload parameters
Reload parameters for TCPREVT
Reload parameters for TCPREVT
Reload parameters for TCPREVT
Reload parameters for TCPXEVT
Reload parameters for TCPXEVT
Reload parameters for TCPXEVT
Reload parameters for TCPREVT
Input config #1
Systematic and parities #1,3,5,...
Interleaver (optional) #1,3,5,...
NULL
Input config #2,4,6,...
Systematic and parities #2,4,6,...
Interleaver (optional) #2,4,6,...
Hard decisions #1
NULL
Output params (opt.) #1,3,5,...
Hard decisions #2,4,6,...
Output params (opt.) #2,4,6,...
Hard decisions #3,5,...

Figure 8. Method 2: paRAM Entries for Standalone

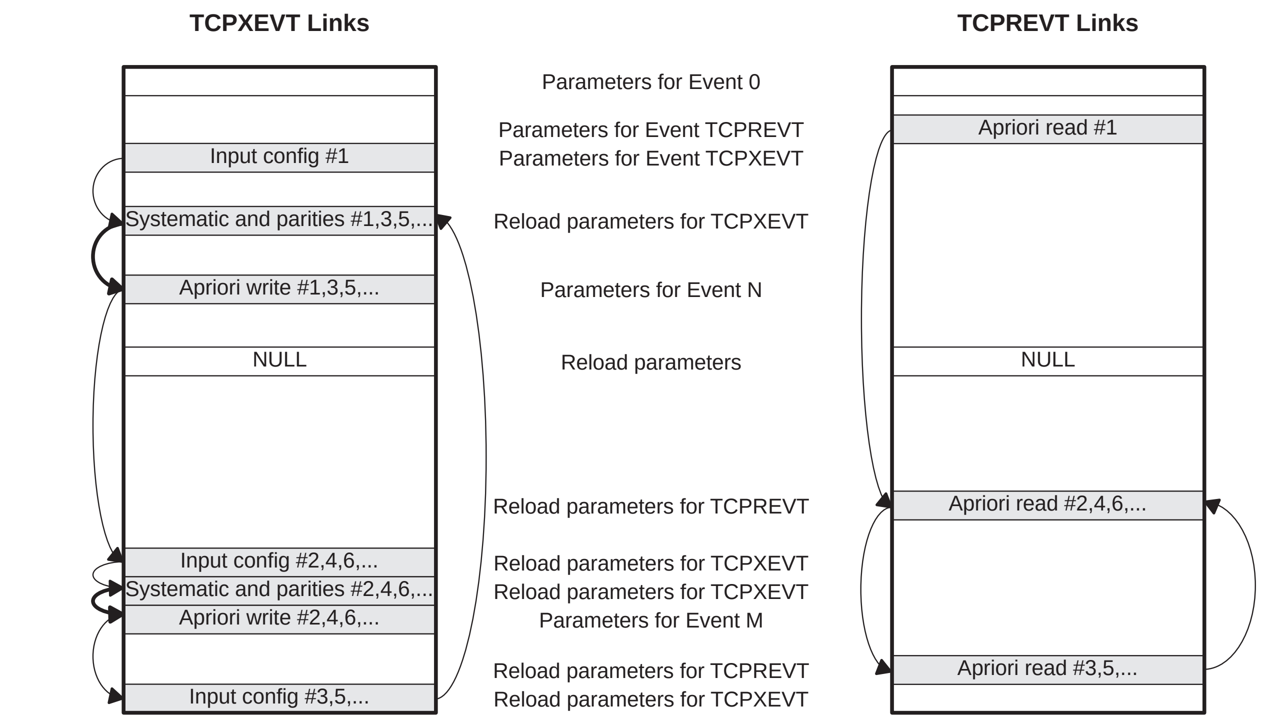
TCPXEVT Links
TCPREVT Links
Parameters for Event 0
Parameters for Event TCPREVT
Parameters for Event TCPXEVT
Reload parameters for TCPXEVT
Parameters for Event N
Reload parameters
Reload parameters for TCPREVT
Reload parameters for TCPXEVT
Reload parameters for TCPXEVT
Parameters for Event M
Reload parameters for TCPREVT
Reload parameters for TCPXEVT
Input config #1
Systematic and parities #1,3,5,...
Apriori write #1,3,5,...
NULL
Input config #2,4,6,...
Systematic and parities #2,4,6,...
Apriori write #2,4,6,...
Input config #3,5,...
Apriori read #1
NULL
Apriori read #2,4,6,...
Apriori read #3,5,...

Figure 9. Method 2: paRAM Entries for Shared Processing (With Previous Context)

The method could be expanded to more than two preprogrammed frames, provided there is space for additional links.

6.3 Method 3: Lowest CPU Interrupt Rate

In this method, we build on the idea of keeping the coprocessor continuously running. In addition to that, the process of EDMA link programming into paRAM is automated: instead of CPU pre-programming a small number of frames into paRAM, the CPU could program a large number of links and temporarily store them in L2 memory. The EDMA is then responsible for transferring preprogrammed links into paRAM.

For this scenario, two additional links in the paRAM are required: one which copies links for even frames from L2 memory to paRAM, and another one which copies odd frames from L2 memory to paRAM. Note that the copy transfer copies not only the links required for the coprocessor, but also the copy link itself, since the source address of prepared links will be different than the previous source address. In order to prevent the copy link from overwriting itself, it can not be stored at the location of Event parameters for the chained channel. Instead, the Event parameters will contain a dummy link which only triggers the actual copy link, stored in Reload parameters.

The scenario is illustrated in Figure 10.

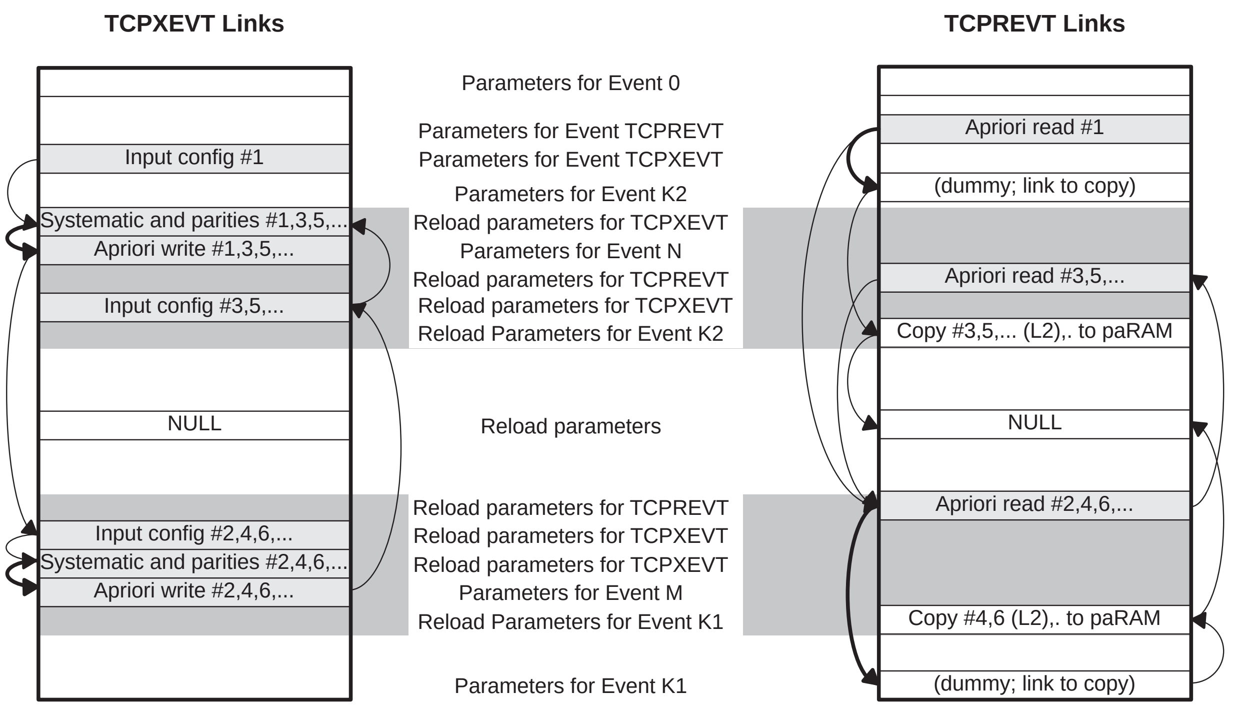
TCPXEVT Links
TCPREVT Links
Parameters for Event 0
Parameters for Event TCPREVT
Parameters for Event TCPXEVT
Parameters for Event K2
Reload parameters for TCPXEVT
Parameters for Event N
Reload parameters for TCPREVT
Reload parameters for TCPXEVT
Reload Parameters for Event K2
Reload parameters
Reload parameters for TCPREVT
Reload parameters for TCPXEVT
Reload parameters for TCPXEVT
Parameters for Event M
Reload Parameters for Event K1
Parameters for Event K1
Input config #1
Systematic and parities #1,3,5,...
Apriori write #1,3,5,...
Input config #3,5,...
NULL
Input config #2,4,6,...
Systematic and parities #2,4,6,...
Apriori write #2,4,6,...
Apriori read #1
(dummy; link to copy)
Apriori read #3,5,...
Copy #3,5,... (L2),. to paRAM
NULL
Apriori read #2,4,6,...
Copy #4,6 (L2),. to paRAM
(dummy; link to copy)

Figure 10. Method 3: paRAM Entries for Shared-Processing Mode

References

1. C. Berrou, A. Glavieux, “Near Optimum Error Correcting Coding and Decoding: Turbo-Codes,” *IEEE Transactions on Communications*, Vol. 44, No, 10, October 1996, p.1261−1271.
2. L.R.Bahl et al., “Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate”, *IEEE Transactions on Information Theory*, March 1974, p. 284−287.
3. P. Robertson, P. Hoeher, E. Villebrun, “Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding,” *European Trans. on Telecommunications*, vol. 8, pp.119−125, Mar./Apr. 1997.
4. *Turbo Decoder Coprocessor User’s Guide* − Literature number SPRU534
5. *TMS320C6000 Peripherals User’s Guide* − Literature number SPRU190D
6. 3G TS 25 212 V3.1.0 (1999−12), *Multiplexing and channel coding (FDD)*
7. “Physical Layer Standard for cdma2000 Spread Spectrum Systems,” *TIA/EIA/IS-2000-2, prepared by TR45.5*.

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265