A 2.5 mW - 10 Mbps, Low Area MAP Decoder

Gord Allan, M.Sc (Eng)

Decoder Description and Specifications

An IC Engines project in association with Carleton University.

Abstract

A VLSI implementation of a modified log MAP algorithm is detailed. The particular design's core, characterized and annotated with TSMC's 0.18u standard cell library, can process data at up to 60 Mbps, consuming only 15 mW of power and 24000 gates.

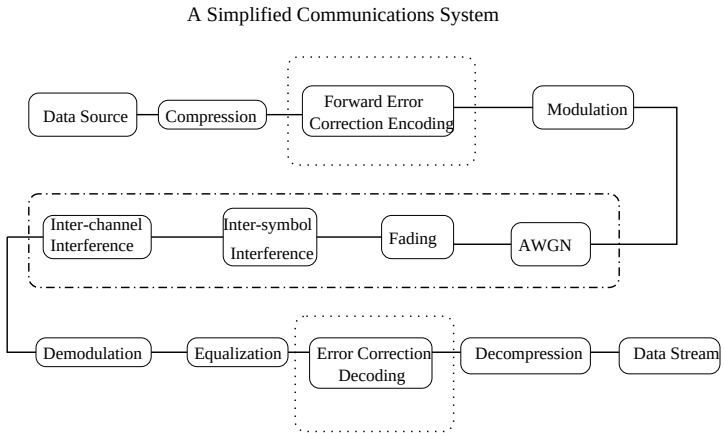
Table of Contents

List of Figures

1.0 Introduction

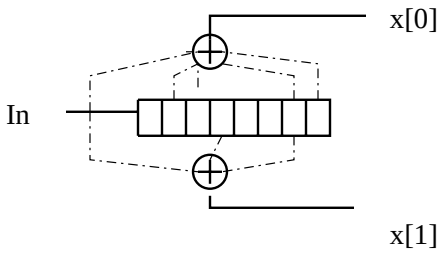
A general communications system is shown in Figure 1. Although all of it's components are of vital importance, often the most complex, in terms of data processing, is the error correction decoding.

FIGURE 1. A Communications Channel



Forward error correction (FEC) seeks to reduce transmission errors by adding controlled redundancy to the transmitted symbols. Depending on the channel and system properties, different error-correction schemes are employed. For the wireless channel, until recently the best approach has used convolutional coding at the core of the transmitter (as shown in Figure 2). The input data is hashed with the previous K-1 bits, where K is the constraint length of the code. The symbols sent over the channel are then a function of the current bit, and the previous bits. This hashing has the effect of spreading the bit information over multiple periods, and thus adds more protection to the data. The longer the memory length, the stronger the code becomes.

FIGURE 2. A K=9 UMTS Convolutional Encoder



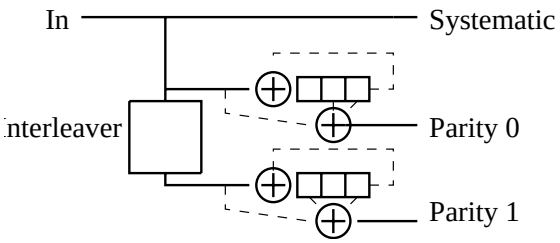
K=9 Convolutional Encoder
Upcoming 3G Standard
Requires 256 State Viterbi based Decode

The computational complexity is in the receiver, where typically a Viterbi decoder is tasked with recovering the original data bits. The algorithm (see Section 2.1), models every possible state of the transmitter and computes the various input probabilities. Therefore, for every added bit in the transmitter, the decoder complexity doubles. The upcoming standard for 3G mobile communications in Europe calls for a constraint length of 9 [1], which requires 256 decoder operations per bit. The search has been on for quite some time to reduce the complexity of the Viterbi decoder. The most advanced implementation found to date (June 2000), is reported by Chang, Suzuki and Parhi in [2]. In 0.5µ technology, by scaling the voltage down to 1.8V they managed to attain a 2 Mbps 10 mW 256 State Viterbi decoder.

Turbo codes, introduced in 1993 by Berrou, Glavieux, and Thitimajahima [3] have unprecedented performance in terms of coding gain at the expense of receiver complexity. Turbo codes use similar convolutional techniques, with much shorter memory lengths, and perform decoder iterations. The output of the first decode step is used in a 2nd iteration, et. cetera. Due to its phenomenal, near Shannon limit decoding, turbo codes are another option specified for the 3G UMTS standard, with the encoder shown in Figure 3.

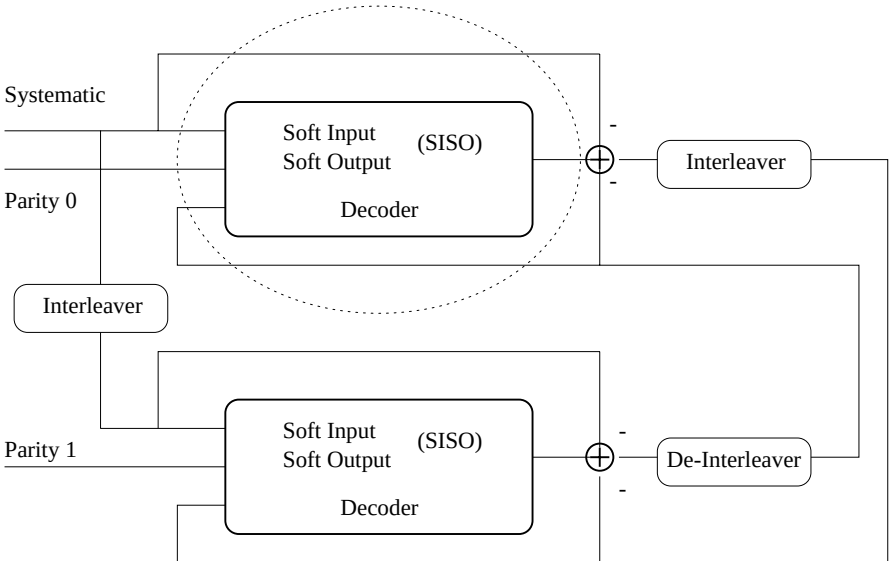
The turbo encoder of Figure 3 consists of two offset convolutional encoders of constraint length 4. For every input data bit, 3 bits are sent over the channel. The code rate is therefore 1/3, although the code can be punctured to reduce the required channel bandwidth. The standard puncturing form sends $\text{Systematic}_{\text{bit0}}$, $\text{Parity0}_{\text{bit0}}$, $\text{Systematic}_{\text{bit1}}$, $\text{Parity1}_{\text{bit1}}$, $\text{Systematic}_{\text{bit2}}$, $\text{Parity0}_{\text{bit2}}$, et. cetera. In this case, the code rate is raised to 1/2, where every 2nd parity bit has been ignored. This is easily compensated for in the receiver by “stuffing” nulls into the channel information in where the punctured bits would occur.

FIGURE 3. Encoder for Parallel Concatenated Convolutional Codes (Turbo Codes)



The turbo decoder (Figure 4) begins by treating each of the two convolutionally encoded streams individually. Each decoder takes in the systematic data and its respective parity sequence, eventually producing two separate estimates of the decoded data bits. These new data estimates are then swapped and treated as the systematic inputs for the other decoder in another iteration. With successive iterations, the estimated data streams converge and the result is taken[1].

FIGURE 4. Iterative Turbo Decoder Structure



Each convolutional decoder needs to support a K=4 code and thus models 8 states in the transmitter. This is a significant complexity reduction over the K=7 code of current standards which must model 256 states in the transmitter. Its use in a turbo decoder however, requires that it produce confidence levels for each output bit. The Viterbi algorithm

1. Typically after 8 iterations no further gains are achieved. CRC techniques can be employed to stop the iterations once the proper decoded stream has been recovered.

traces back only through the most-likely symbol path, and contains no support for any information regarding the confidence of its decisions.

The constituent decoders take in quantized data from the channel, and output quantized confidence levels for each decoded bit. They are therefore termed soft input, soft output (SISO) decoders. Although soft input decoders are common, soft outputs are more difficult to accommodate. In decreasing order of accuracy and complexity, algorithm options for implementing this type of decoder include the Maximum A-Priori (MAP) / Log-MAP, Max-Log-Map, and Soft Output Viterbi (SOVA). A very good tutorial overview of turbo codes and decoder structures is presented by Woodward and Hanzo[4].

This implementation conglomerates numerous architectural novelties to implement an extremely low area, low latency, power efficient Log-MAP decoder for use in an iterative turbo decoder structure. The MAP (Maximum A-Priori) algorithm and its logarithmic counterpart (Log-MAP) are optimal, with their only imprecision due to rounding errors. On the other hand, the Max-Log MAP algorithm performs about 0.2 dB worse, and the SOVA algorithm 0.7 dB worse (measurements at a bit error rate of 10^{-4}) [4]. The former algorithms, however, typically benefits from lower receiver complexity.

2.0 Trellis Decoding

Trellis decoding algorithms all consist of forward and reverse analysis of the data stream. The Viterbi algorithm maintains the same forward calculations as other more optimal decoders, but simplifies the reverse path such that only the maximum likelihood output is considered. While this is normally all that is desired, the Viterbi algorithm is ill-suited to produce soft outputs.

The mathematical preliminaries of the MAP and Viterbi algorithms (VA) are covered in [4] and elsewhere, but are skipped in this conceptual analysis. Rather, an intuitive description follows.

The MAP and Viterbi algorithm (VA), model each possible state of the transmitter in a trellis (Figure 5). Given the received stream, a probability is associated with every possible state transition. Half of the possible transitions correspond to a 1 input, the other to a 0 (or -1 in BPSK) input. The option with the highest probability is taken as the winner.

Assume for the moment, that the decoder has available the entire frame of received channel data. Due to the transmitting encoder, only certain logical sequences of parity and systematic bits are valid. To compute the probabilities of the individual input bits, all algorithms break the computations into three distinct calculations:

1) Given the previous channel history at time k, what is the probability (α) that the decoder is in a particular state (s').

2) Given the input symbol (quantized systematic and parity bits) at time k what is the probability (Γ) of having made a transition from state s' to state s, for all (s',s).

3) Given the future sequence of bits, what is the probability (β) that we are in state s at time k+1.

The final output probability is a comprise between the three terms.

This regression is possible, with the initial knowledge that the transmitter begins and ends a framed sequence in state 0. The significant computational advantages of this decoder come about by reducing the requirements of step 3, as covered in section 2.2.

2.1 Worked Example (α and Γ Calculation)

Operation of the MAP and Viterbi algorithms are clarified through a worked example. Referring to Figure 5, the transmitter begins in state 000_{binary}. The receiver, knowing that the transmitter begins in this state, ignores all the other possibilities for the timebeing. From state 000, the only two possibilities for the next state are 000 (for a 0 input) and 100 (for a 1 input). For each of these two options, the receiver would expect a logical (Systematic=0, Parity=0) pair or a (Systematic=1,Parity=1) pair.[1]

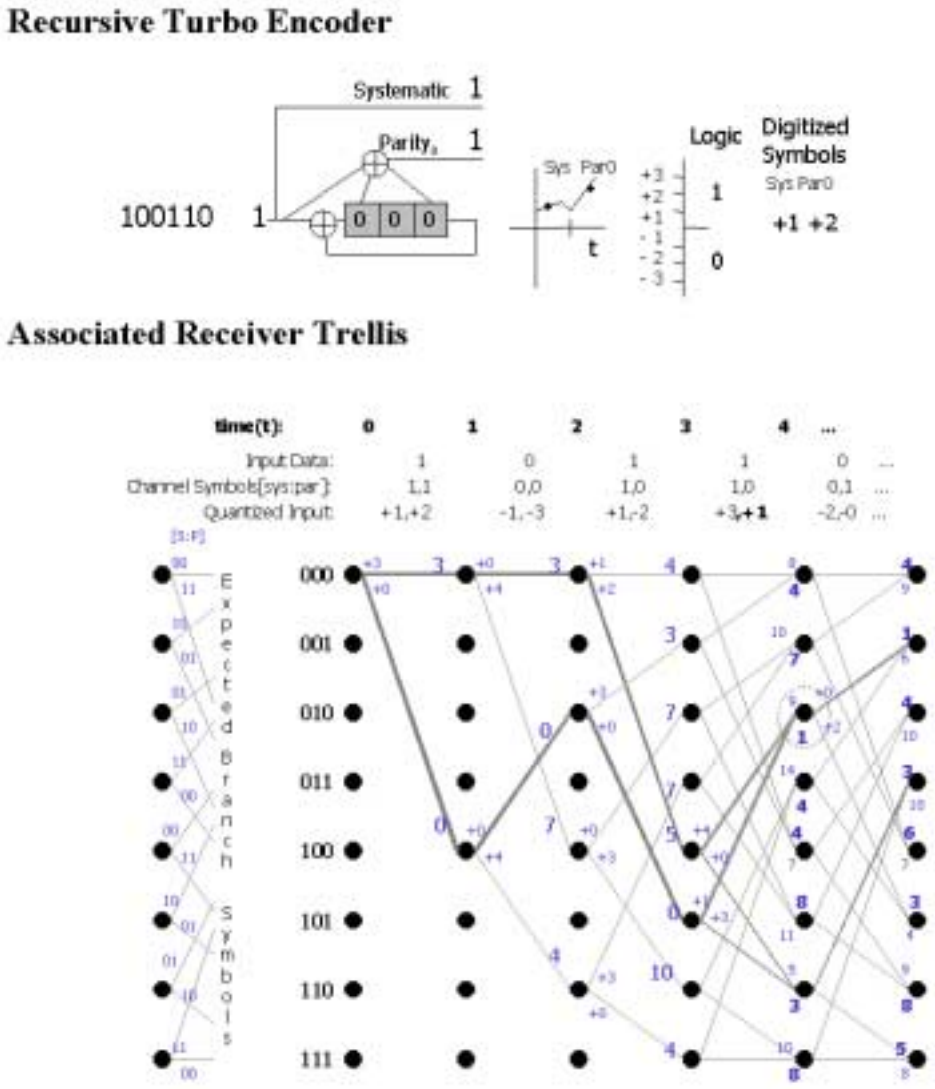
After quantization of the noisy channel inputs, the receiver takes the results as +1 and +2 (given a quantization range from -3 to +3). This indicates that the (1,1) pair was likely transmitted, and therefore the most likely state becomes 100. To track the relative probabilities of each state, paths that agree with the received symbols can be credited, or those that disagree can be penalized[2]. By convention, throughout this design it has been chosen to penalize paths that disagree with the channel symbols. Associated with state 000 is a penalty of 3 (since the quantized inputs disagreed by +1, +2 from the expected values.) So far, state 100 has no associated penalty. Those states which are not possible can be assigned a very large penalty to begin the frame.

The penalties associated with each transition are, by convention, called branch metrics and are logarithmically proportional to Γ in the above expression. Those penalties associated with a particular state and time are path metrics and are logarithmically proportional to α.

As time progresses and further symbols enter the decoder, more metrics are calculated and each state takes on a relevant path metric to represent its new probability.

1. Systematic is a term used for the original data bit which has been sent over the channel.
2. Common practice has been to both penalize paths that disagree, while crediting those that agree. Since only differences are relevant, both operations are not necessary and is one of the optimizations in the work which significantly reduces its complexity.

FIGURE 5. Example Recursive Encoder and Forward Metric Calculation



At time t=4, there are two possible paths that reach each node. This is of ultimate importance in differentiating between the decoding algorithms. Examining state 010 at time t=4 can provide valuable insight. The most likely path (as transmitted - 1011) has been assigned a penalty of 1, due to the small channel error between times 3 and 4. Its competitor is the only other possible input stream that could have resulted in this state and corresponds to a stream of 0010. It has been assigned a cumulative penalty, or path metric ($\alpha'=\alpha+\Gamma$), of 9. The relative value of each path metric compared to all others in the trellis determines its likelyhood up to this point.

In the Viterbi algorithm (VA), the state takes on the value of the lowest contender, storing only which branch won the decision, and moves on. The soft-output VA (SOVA) is similar, but stores the difference in (two's complement) of the contending metrics to provide some confidence about its decision to throw out a path. In the Max Log MAP algorithm, we similarly throw out the less probable option, but must store the full path metrics (generally 7-8 bits) for each state and time. The Log-MAP and MAP algorithm adjust the path metric slightly to take into account how close the competing paths are. If the competing path metric is far from the winner, the approximations of Max Log MAP and Viterbi hold very well. If the two contending paths into a state are approximately equal, that state should be credited. Woodward covers the distinction quite well.

To clarify mathematically, while the Max-Log MAP and Viterbi algorithms take only the best of two contending paths into a state, the Log MAP and MAP algorithms are ideal in that they factor in the probabilities of both contenders. The ideal metric is shown in eq. 1 [4].

$$\text{Path Metric} = \min(\text{met0}, \text{met1}) - \ln(1+e^{-|\Delta|}) \quad \textbf{(EQ 1)}$$

For large Δ, the Path Metric = min (met0, met1) and the approximation of Max Log MAP and Viterbi hold. The Log-Map algorithm is well known to perform approximately 0.2 - 0.3 dB better than the Max Log MAP algorithm [4].

2.2 Reverse Metric (β) Calculation, Outputs and Windowing

From 2.1, we have calculated all of the forward metrics (α's) which include the required information about α and Γ from the previous time step. If we know that the receiver ended the frame in state 0, we can calculate the reverse metrics (β) for each symbol, from the last bit to the first, using the same recursion of 2.1. Note that the reverse regression must have the trellis re-wired to correspond to travelling in the opposite direction.

Given the previously computed, and stored α's, the expression for the output bit probability is given by:

$$\text{LLR(bit)} = \ln(P[1]/P[0])$$

$$\text{LLR(bit)} = \ln(P[1]) - \ln(P[0])$$

$$\text{LLR(bit)} = \min_{\text{over 0 transitions}}(\alpha + \beta + \Gamma) - \min_{\text{over 1 transtions}}(\alpha + \beta + \Gamma) \quad \textbf{(EQ 2)}$$

Where the log domain is used for two main advantages. First, it allows us to use only addition and subtraction elements to compute α, β and Γ versus complex multiplications in the non-logarithmic domain. Second, it provides an extremely large dynamic range for our output probabilities as shown in Table 1.

TABLE 1. Conversion between LLR and Probability.

LLR	P[1] = $e^{llr} / (1+e^{llr})$
21	0.9999999992
...	
4	0.982
3	0.952
2	0.880
1	0.731
0	**0.5**
-1	0.269
-2	0.120
-3	0.048
-4	0.018
...	
-21	0.0000000007

In order to store the necessary forward metrics (α's) across the entire frame requires an excessive amount of hardware. For example, in an 8 state decoder, if each α requires 7 bits and a frame is 512 bits long, necessary metric storage is:

$$\text{Metric Memory Size} = \text{number of states} * \text{metric width} * \text{frame size}$$

$$\text{Metric Memory Size} = 7*8*512 = 28 \text{ kbits} \quad \textbf{(EQ 3)}$$

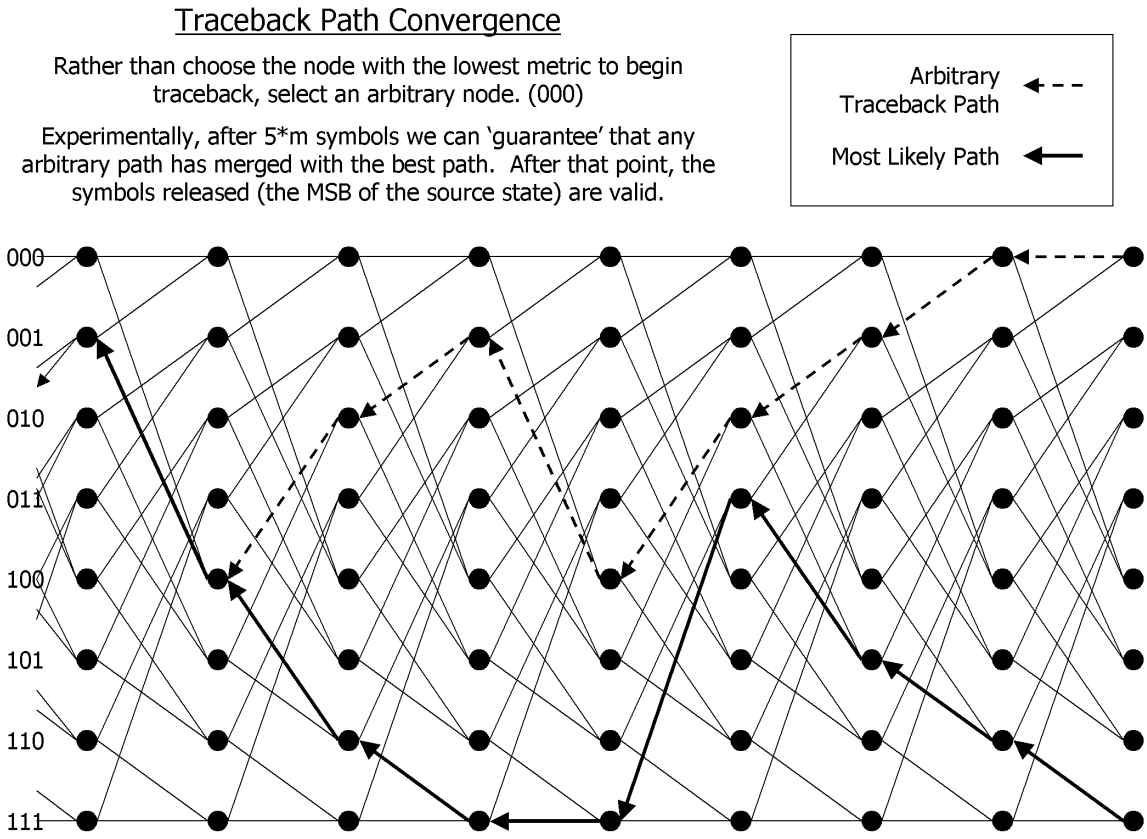
As a first estimate using only D latches to store this amount of data (without C/L) would consume 171k gates. For comparison, this decoder implementation has a total gate count of 24k. Other reasons also make this approach unattractive in that, a) it has an output latency of 2*frame length - which is unacceptable in almost all applications, and b) the amount of required hardware is dependent on the maximum frame size, and c) the channel information must also be stored for the entire frame, implying a channel memory size of:

$$\text{Channel Memory Size} = \text{quant bits} * \text{frame size} * (\text{bits/symbol} + 1_{\text{a-priori}})$$

$$\text{Channel Memory Size} = 3*512*3 = 4.6 \text{ kbits} \quad \textbf{(EQ 4)}$$

In the Viterbi algorithm it has been widely shown that instead of waiting for the entire transmitted frame to be received (required to compute the future oriented term of the analysis), it is possible to start the reverse regression for each bit some distance (W) away from time k, with minimal to no degradation in performance (Figure 6). This is because the most likely path will have converged, at some point, with any arbitrary path by the time it reaches the decision point.

FIGURE 6. Illustrating Traceback Path Convergence



This allows data to be streamed, rather than arranged in distinct frames. More importantly, this observation significantly reduces storage requirements in each of the algorithms since we must now store only W bits of data ahead of time k. In the MAP algorithms, this distance is referred to as a window size, whereas in Viterbi it is called the traceback length. The value for W is empirically determined to be 5 * the memory length of the transmitter.

In MAP, a sliding window has typically been used With this technique, for each α calculation we re-evaluate the reverse metrics from time k+W down to k. Hence, W β calculations are made per α calculation, necessitating a high complexity decode.

The modified Log MAP decoder designed here uses a fixed window approach, which is responsible for much of the power savings. We employ two pipelined reverse β units. One of which starts from a random state ($\beta_{initials}$ are undefined) and, using the channel data from t=W to t=2W, performs an internal traceback to initialize the β values to a valid starting point. The second traceback unit uses this starting point, and calculates and stores the valid β values to a metric memory. As the forward metric moves through

to calculate its α's, the associated β's have already been calculated. They are read from the metric memory and all of the information is available to produce valid output LLRs as given in equation 2.

Some justification of the fixed windowing technique is in order. The sliding window technique, shown to be valid [5], ensures that all outputs consider W bits into the future. The argument against the fixed window, is that while a particular bit at time k considers 2*W bits into the future, the output for bit k+W considers only W bits into the future. If bit k is more reliable than bit k+W, then the fixed window technique is invalid. But, by the validity of the sliding window technique, we know that further information beyond bit W into the future is irrelevant to our decision. Therefore, although bit k sees twice the necessary amount into the future, it is no more reliable than bit k+W and the fixed window technique is valid.

Using fixed windowing then, we only perform 2 β calculations for every α calculation. One of the β units is searching for an initialization value, which is then used by the other unit to follow through and calculate all of the valid β's.

The complexity of this technique then, is only 3 times that of the forward recursion of the Viterbi algorithm. Also, using a well designed addressing technique, it is only required to store W words of metric data - or 7*8*16 = 896 bits in this implementation.

3.0 Log Likelyhood Scaling and Metric Range

To limit storage and computation requirements, it is desirable to use the fewest bits for representation wherever possible. This is especially true in the case of the path metrics, since the bulk of storage is required to accommodate the reverse metrics for each state in the trellis.

As the forward and backward recursions progress through the trellis, the metrics grow unbounded. Since only the difference between the metrics is of importance, they can be re-normalized at any time.

It has been shown[5], that for a constraint length K code with possible branch penalty μ, the maximum dynamic range between any two metrics at one time is (K-1)*μ. In this decoder, 4 bits of quantization are used for each of the channel bits and the incoming LLR. A maximum branch penalty of 3*7=21 can therefore be applied to any branch. For the K=4 code, the maximum difference between any two metrics in this trellis is 3*21=63, requiring 6 bits.

If at each step we determine the minimum metric, and subtract it from the others, normalization is accomplished. This incurs a penalty however in that it becomes necessary to add a comparator and extra subtraction step for every forward and reverse computation. Typically, metric re-normalization is best performed by adding an extra bit to the metric widths, and employing one of a few techniques. One option is to perform a threshold detect, and constant subtraction - which is much cheaper in hardware than a full comparison and variable subtraction. Another, less intuitive option, allows the metric to overflow. Provided the compares in the trellis are performed with two's complement subtractors, proper operation is still achieved. This is the technique used in this

decoder since a two's complement subtract is already required to apply the Max-Log MAP correction factor. The metrics in the design have therefore been set to 7 bits each. It has also been chosen to only penalize branches which do not agree with the received channel symbols. Smaller metrics therefore correspond to more likely paths.

Given a maximum applied branch penalty of 21, the maximum output LLR is + – 42. For initial decoder iterations where we are not very confident about our outputs, we can trade high precision for lower-dynamic range (e.g. from an LLR of -3 to 3, precision of 0.1), whereas for subsequent iterations, we prefer higher dynamic range at the cost of lower precision.(eg. LLR from -84 to 84, precision of 4). In this design, this is accomplished through the use of a scaling factor. The scaling factor varies, in multiples of 2, from 1 to 32. While the internal representation is unchanged, the decoder needs to know what the values of the LLR are in real terms in order to apply the Max-Log Map correction factor appropriately. This factor is responsible for the ideal performance of the Log MAP algorithm of the Max-Log MAP algorithm.

TABLE 2.

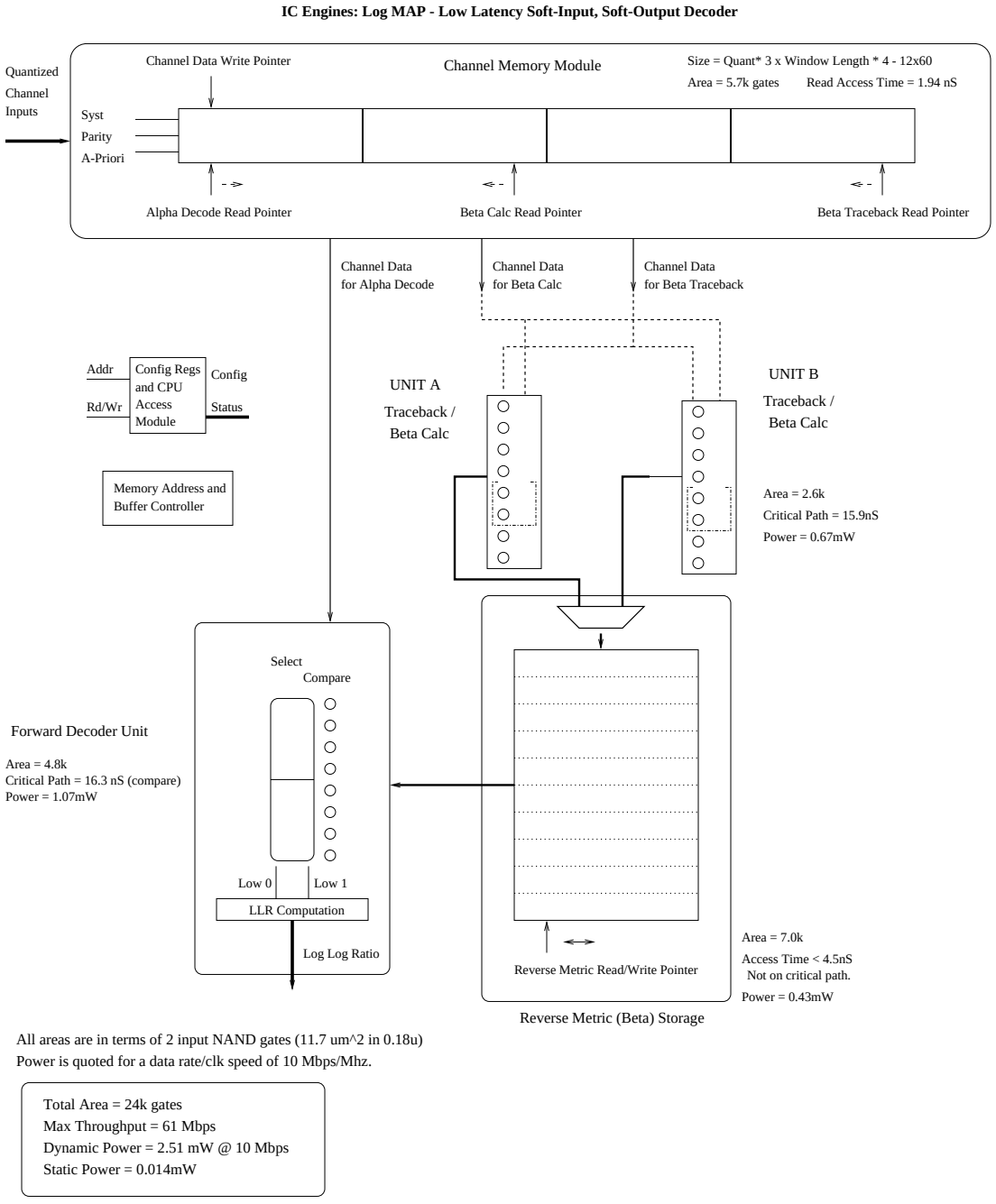
LLR[2:0] least sig bits	Hardware Represent.	Scaling Factor=1	Scaling Factor=2	Scaling Factor=4	Scaling Factor=8	Scaling Factor=16	Scaling Factor=32
000	0	0.000	0.00	0.0	0	0	0
001	1	0.125	0.25	0.5	1	2	4
010	2	0.250	0.50	1.0	2	4	8
011	3	0.375	0.75	1.5	3	6	12
100	4	0.500	1.00	2.0	4	8	16
101	5	0.625	1.25	2.5	5	10	20
110	6	0.750	1.50	3.0	6	12	24
111	7	0.875	1.75	3.5	7	14	28
MAX=6 bits	**Range -42 to 42**	**LLR=-5.25 to 5.25**	**LLR=-10.5 to 10.5**	**LLR=-21 to 21**	**LLR = -42 to 42**	**LLR = -84 to 84**	**LLR=168 to 168**
Precision	**1**	**0.125**	**0.25**	**0.5**	**1**	**1**	**4**

In order to apply this correction factor, Woodward [4] and others propose a small look-up table of 8 words, which store the value of $\ln(1+e^{-|\Delta|})$ for Δ in the range of 0 to 5. The resultant correction terms range from 0.693 to 0.006. Since the number of metric bits limits the precision of the path metrics (Table 2), the correction factor has no effect for scaling factors above 8. Below scaling factors of 8, where precision is available, the correction factor is applied using a combinational circuit rather than a LUT. Both techniques were synthesized and the LUT was determined to be more expensive in terms of both power and area.

4.0 Fixed Window - Log MAP Decoder Implementation

The simplified system diagram of the MAP decoder is shown in Figure 7.

FIGURE 7. Log MAP Decoder - System Diagram with Power, Timing and Area Annotation



The systems primary IO signals include:

Inputs: Channel Data - Quantized Systematic bit
- Quantized Parity bit
- A-Priori information from previous decoder iterations

Output: Log-Log Ratio (LLR) - latency of 65 cycles after the input bit

For control, the system has only 4 inputs:

clk - System clock (< 60 Mhz, Duty Cycle > 3%)
reset - system reset (apply for over 1 cycle), remove synchronously
enable - a low pauses the circuit operation, and optionally gates the clock
sync - an optional input pulse resets the status counters on the first bit of a frame

A CPU interface is optionally provided which allows asynchronous read monitoring of many internal circuit nodes and other status information. It can also be used to reconfigure the decoder for different encoder configurations and to change the output scaling factor. To reduce IO count, area, net loads, and wiring density, the CPU read functionality (a large multiplexor array) can be eliminated after successful prototyping. It is therefore not included in the power and area analysis.

Throughout the implementation power estimates are derived through the use of Synopsys Power Compiler. After appropriate signal annotation of the known nodes of the design, it calculates the intermediate activity factors, and using the library models, calculates the power consumption. Given good annotation, Synopsys claims the tool to be 'relatively' accurate. It should be noted that this power analysis does not take into account glitching, interconnect capacitances, or IO pad power.

Critical Statistics:

- Area = 276883 um^2 = 23.66k gates
- Power @ 10 Mhz = 2.51 mW
- Critical Path - Pipelined Comparator operation: 16.3 ns
  2nd longest path - Forward/Rev Metric Calculation: 15.9 ns
- Max Clock Frequency/ Symbol Rate = 61 Mhz/Mbps

As eluded to throughout Section 3, the required modules include:

- Forward Calculation Unit (w/ 8 Add Compare Select (ACS) Units + 2 Comparators)
- 2 Reverse β Calculation/Traceback Units (w/ 8 ACS Units each)
- β Metric Memory storage Unit
- Channel Memory storage Unit
- Control Unit
- CPU Configuration and Status Monitoring Unit

The system timing and module descriptions follow.

FIGURE 8. MAP Decoder System Timing

Interface Timing of the MAP Decoder

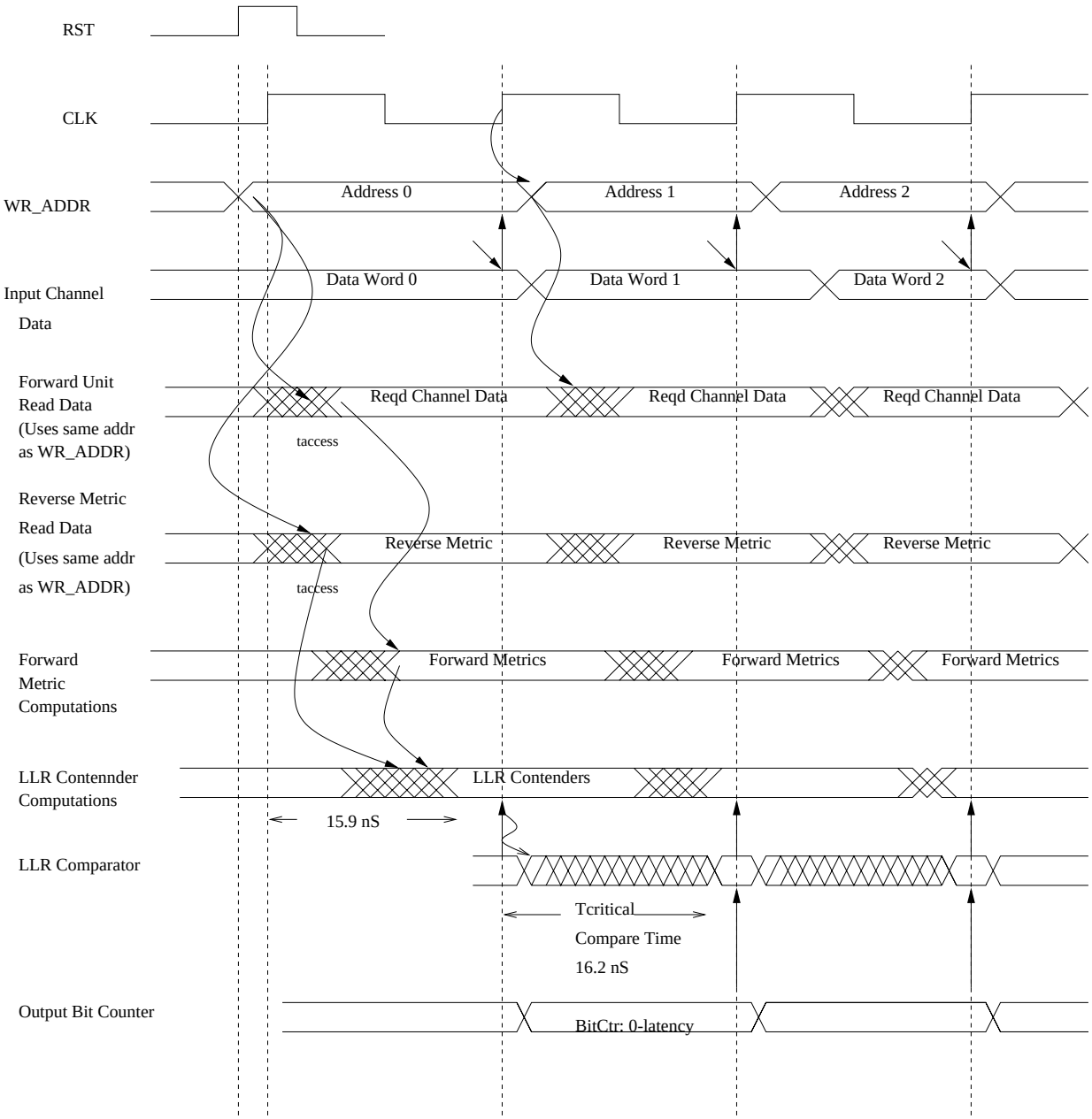
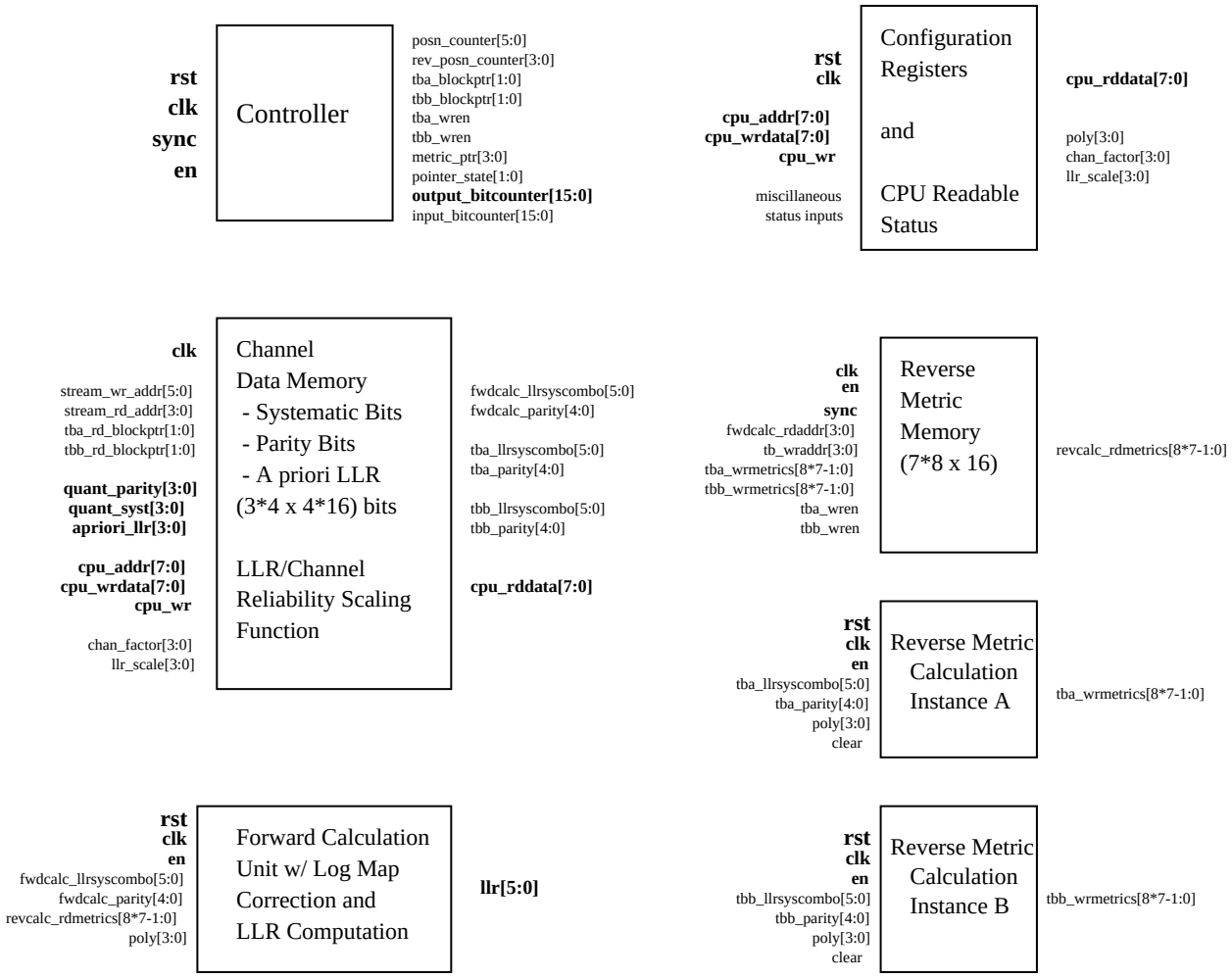


FIGURE 9. Module IO connections

Detailed Signal Definitions and Module Interaction



Notes:

Bold signal names represent pimary IO pins.
The CPU interface is not essential to circuit operation but allows status monitoring and re-configuration.
The en signal acts to "pause" the decoder operation.
The sync signal can optionally be used to reset the output bitcounter on the first bit of a new frame.

4.1 Forward α Calculation Unit

Given:

- previously computed reverse path β's
- channel information (parity and combination of a-priori and systematic bits)
- internal: α from previous time step

Computes:

- α' for next time step (with log map correction factor)
- Γ for each branch transition
- scaled output LLR as $\min_{\text{over 0 transitions}}(\alpha + \beta + \Gamma)$ $- \min_{\text{over 1 transtions}}(\alpha + \beta + \Gamma)$

Critical Statistics:

- Metric Width = 7 bits
- Input Quantization: Parity/Symbol - 4 bits sign-mag, a-priori 5 bits sign-mag
- Normalization Scheme: Increased metric width, allowable
- Metric Comparator: 2's complement subtraction
- Area = 55935 um^2 = 4.78k gates
- Power @ 10 Mhz = 1.07 mW
- Critical Path - Pipelined Comparator operation: 16.3 ns
  2nd longest path - Forward Metric Calculation: 15.9 ns

The forward calculation is mainly comprised of 8 Add-Compare-Select (ACS) units and 2 comparators.

The ACS units, one for each state, are responsible to compute its respective α, taking into account the max-log to log MAP correction factor and scaling. Rather than the traditional approach of branch metric(Γ) calculation, where each ACS is tasked with determining the appropriate branch metric, the top level forward calculation unit computes all possible options and feeds these appropriately to the sub units. Doing so improves both power and area, with no additional speed penalty. The ACS units each contain combinational logic to implement the max-log MAP to log MAP correction. Synthesis was carried out using both this technique and a static LUT, and the area and power of the C/L approach was superior (8% less area, 15% less power).

Two comparator options were also evaluated. One was coded behaviorally as a standard form verilog comparator, the other structurally. The structural implementation relied on a recursive, somewhat asynchronous technique, where each contender would eliminate itself once it found it was too large. After synthesis, although having approximately the same area and power results, the structural approach was, at worst case 3 times slower than the simple comparator function.

4.2 Reverse Calculation Units (x2)

Given:

- channel information (parity and combination of a-priori and systematic bits)
- internal: β from previous time step

Computes:

- β' for next time step (with log map correction factor)

Critical Statistics:

- Metric Width = 7 bits
- Input Quantization: Parity/Symbol - 4 bits sign-mag, a-priori 5 bits sign-mag
- Normalization Scheme: Increased metric width, allowable
- Metric Comparator: 2's complement subtraction
- Area = 31675 um^2 = 2.71k gates
- Power @ 10 Mhz = 674 uW
- Critical Path - $4.52^{tmemaccess}$ + 11.35 = 15.9 ns

The Reverse calculation is equivalent to the forward one, but the ACS units are interconnected in a different pattern. It therefore shares many of the same properties as the forward calculation unit. It does not however, need to contain the compare logic used to create the output symbol, and is thus smaller and more power efficient than the forward array.

Each unit calculates the β's as it progresses through the trellis. While one unit is producing relevant data (that is being written into the metric memory), the other is merely performing the calculations to find a suitable starting state.

4.3 Reverse Metric Storage

Given:

- reverse β's as calculated by one of the β calculation units
- internal: the states of all β's (k to k+W)

Outputs:

- the relevant β to the forward calculation and decode unit
- internally: stores the proper β

Critical Statistics:

- Width: $2^{(K-1)}$ * Metric Width = 8 * 7 = 56 bits
- Number of Words: Window Size (W) = 16

- Size = 56 * 16 = 896 bits
- Area = 82364 um^2 = 7.04k gates
- Power @ 10 Mhz = 433 uW
- Addr -> Read Data access time < 4.52 ns

The reverse metric storage was initially attempted using standard memory cores. This proved horribly inefficient due to the small required memory size. It is also the case that, in order to use the minimum memory size possible, a read must occur from the same address where a write is to be performed. Using a memory, to attempt this would require a 2nd clock, or a relatively complex control scheme combined with an extra word of memory.

Instead, a behavioural register file was implemented. Synopsys proves to be horribly inefficient at compiling such structures, and so a structural approach was taken. The address decode is performed and signal gating is used extensively throughout the storage structure to reduce fan-out and unnecessary toggling. Reads are performed using a multiplexor tree, rather than through the use of tri-states, due to its simpler design complexity.

Using a DFF to store each bit is relatively expensive. Depending on the cell which the compiler chooses, the a DFF ranges in size from 6 to 9 standard gates. Choosing not to include a RST and en signal on the DFFs can mitigate the size. Additional area gains can be achieved by implementing the storage elements as D latches instead. With this implementation, each bit of storage occupies the area of only 4.43 equivalent gates. Using the latch approach, careful consideration was given to prevent glitching on the latch signals, and to evaluation of the timing implications. At 10 Mhz, this approach enforces that the high portion of the clk last at least 3.5 nS. This enforces a clk duty cycle in the range of 3.5% to 96.5%.

Special care also had to be given to the memory addressing of this unit. In order to properly sequence the data using the smallest ram size to address must ‘ping-pong’ back and forth from 0..15, then 15..0 etc. The addressing is done in the control logic.

4.4 Channel Data Storage

Given:

- input information bits (A-priori data, Quantized Symbol and Parity information)

Outputs:

- relevant Symbol and adjusted systematic output to the fwd unit
- relevant Symbol and adjusted systematic output to the rev β traceback unit
- relevant Symbol and adjusted systematic output to the rev β calculate unit

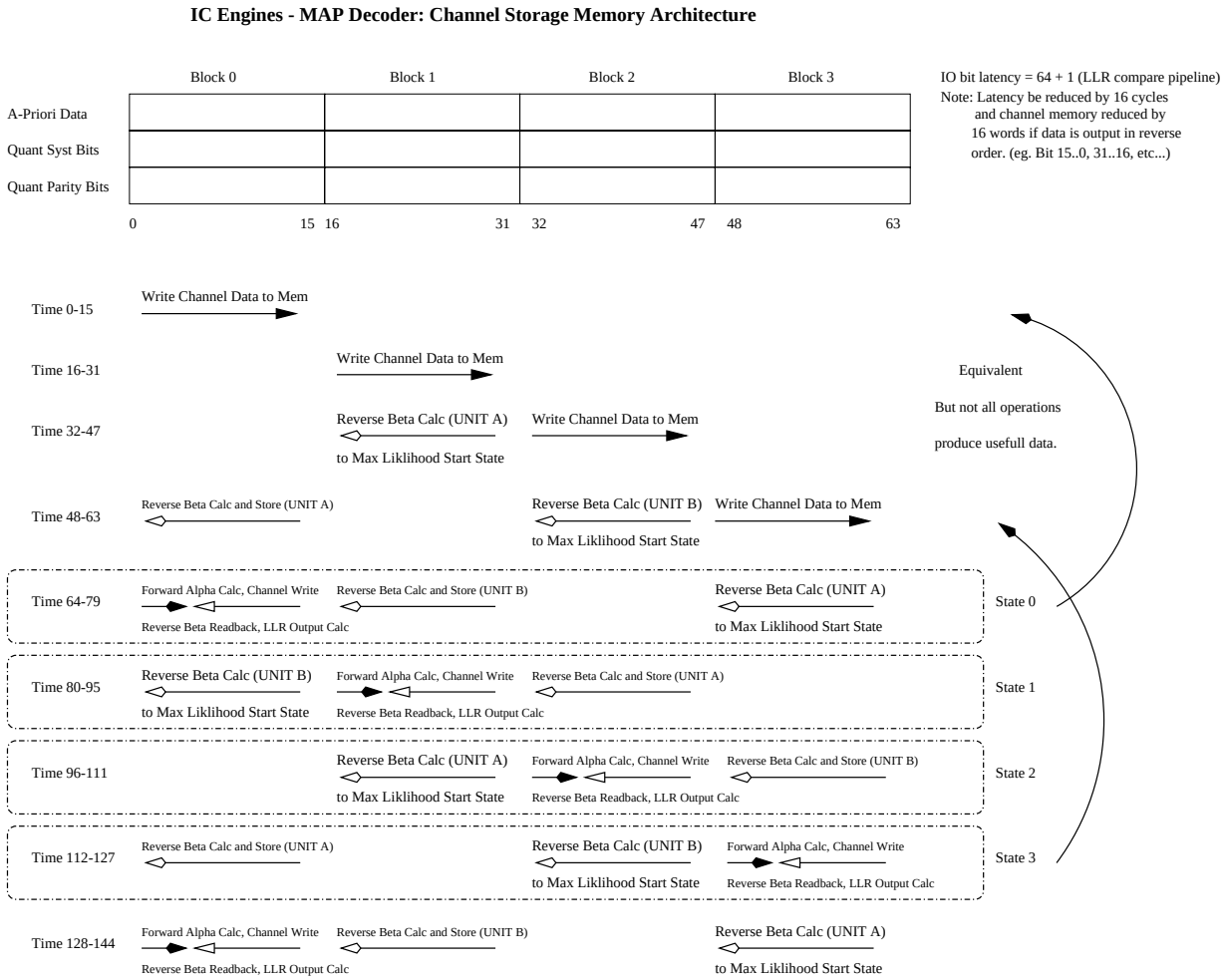
Critical Statistics:

- Quantization: Parity/Symbol - 4 bits sign-mag, a-priori 4 bits sign-mag

- Width: Symbol bits + Parity bits + a-priori bits = 12 bits
- Number of Words: 4*Window Size (W) = 4*16
- Size = 12 * 64 = 768 bits
- Area = 67314 um^2 = 5.75k gates
- Power @ 10 Mhz = 368 uW
- Addr -> Read Data access time = 4.52 ns

Similar to the metric storage unit, the channel storage is also implemented with D latches. The storage unit has 1 write port and 3 read ports. The addressing of which is rather complex, but follows from the description of the algorithms in Section 3. It is detailed in Figure 10. The read addr for to the reverse calculation units always go from 15..0, 15..0, etc. while the rd pointer for the forward decode unit goes from 0..15, 0..15.

FIGURE 10. Channel Memory Access and β calculation co-ordination



As data is being read out for the forward unit, that memory space is no longer needed and can be written to by the newly incoming channel data.

Due to the structure of Figure 10, the decoder latency is 4*W = 64 bits. There is an additional cycle of latency added due to the pipelines compare in the decoder's output stage, bringing the total latency to 65 bits.

It can be shown that we can reduce the channel memory requirements by W, if we allow the bits to come out in reverse order (i.e. from 15..0, 32..16, etc.) This may be allowable, and advantageous, in our system depending on the interleaver structure. With this method, the output latency is reduced to 49 bits. If this technique were employed, we would use the active reverse array to perform the final output decisions, rather than the forward array.

4.5 Control Unit and CPU Access Module

The control unit contains one master counter of 6 bits that revolves from 0 to 63. This is used directly as the channel write address and forward decode address to the channel storage unit.

Derived from this counter through combinational logic are all of the other required signals and address lines - as detailed in the previous sections.

The exception is the input and output bit counters of 16 bits, that are merely informational. The input counters lower 6 bits follow the master counter, while its upper 10 bits are formed with an independent counter. The reverse bit counter provides an indication of what bit is currently at the output of the decoder. It is a separate 16 bit register which follows the input counter - the latency of 65 bits.

The CPU access module is composed of two distinct sections, a read and write module. The write module is used to re-configure the decoder's polynomial and scale settings from their defaults, while the read module provides asynchronous access to many of the internal nodes of the circuit. The read module is only used for debugging and is this not included in any of the power and area analysis.

5.0 Comparison to other Implementations

Unfortunately, there is a conspicuous absence of implementation specifications for soft-input, soft-output decoders [6]. Most of the literature has instead focussed on the architectural trade-offs in unspecific forms. Much of this has focussed on optimizing the number of bits for metric representation, quantization, a-priori data, etc. Other work has evaluated, at an algorithmic level, the computational and memory requirements of Log-MAP vs. MAP vs. Max-Log MAP vs. SISO.

Some hard specifications have been found in various technologies and forms. In the following list, references are provided and attempts are made to scale the design into terms appropriate for comparison.

This Design: IC Engines Log-MAP Decoder

- 1.8V, 0.18 um CMOS technology TSMC standard cells
- K = 4 (8 state), state parallel SISO unit - Fixed Window Log MAP Algorithm
- Core Area = 276883 um^2 = **23.66k gates (11.0 k logic, 12.7 k storage)**
- Power Estimate @ 10 Mhz/Mbps = **2.51 mW**
- Max Clock Frequency/ Symbol Rate = 61 Mhz/Mbps

[11] Bickerstaff, Garrett, Prokop, Thomas, Widdup, Zhou, Nicol, Yan

- 1.8V, 0.18 um CMOS
- K = 4, 8 state Log MAP
- **Area = 85k logic + memory**
- Total core area (including interleaver) = 9mm2 = 769k gates
- the SISO is reused for both decode operations
- Power @ 2 Mbps, 88 Mhz clk,10 iterations = 292 mW(max decode rate)
- The SISO is therefore operating at 40 Mbps
- **SISO Power @ 10 Mbps = 7.3 mW**

[6] Masera, Piccinini, Roch, Zamboni

- A Bit Serial - Log MAP SISO Decoder for use in turbo codes
- CMOS 0.5um technology
- Clock Speed = 50 Mhz, Cycles/Bit = 25, Data Rate = 2 Mbps (10 iter.)
- Maximum clock speed of 94 Mhz
- Area of SISO core = 9 000 000 um^2 ($3mm^2$)
- No Power specifications
- Normalized SISO Area to 0.18 um = Area / $(0.5/0.18)^2$ = **99.7k gates**

[9] Product Specification: Advanced Hardware Architectures Turbo Encoder Decoder

- A commercially available discrete device
- 36 Mbits/sec Max Symbol rate - 2 Iterations, 50 Mhz Clock
- CMOS - unknown feature size
- No core area specifications.
- Current Draw = 250 mA @ 3.3 V, no output loads
- Power = 3.3 * 250mW = 825 mW
- Normalized SISO Power @ 10 Mbps, 1.8 V = Power / $(4*3.6)(3.3/1.8)^2$ = **17.0 mW**

[10] Garrett, Stan

- K=3 (4 state) - Soft Output Viterbi Algorithm SOVA
- CMOS 0.35um

- Area = 0.56 mm^2
- Area scaled to 0.18 um, 8 states = 1.18 M um^2 = **202k gates**
- Max Speed = 7.5 Mbps - Power = 54 mW
- Power @ 1.8V - 0.35um, 3.12 Mbps = 6mW
- **Power scaled to 8 states, 10 Mbps = 38.0 mW**

[8] Suzuki, Wang, Parhi

- K=3 (4 state), Sliding Window MAP algorithm, with early termination
- Max Clock Freq = 32 Mhz, Data rate of 2 Mbps
- CMOS 0.25um
- Core Area = 2.32mm x 1.72mm = 3.99 M um^2
- Scaled Core Area to 0.18um gates = 176k gates (constraint length K=3)
- Scaled Core Area to 0.18um gates = 353k gates (constraint length K=4)
- Transistors: 100k logic, 200k memory - logic:mem area ratio of 2:1
- Scaled **Logic** area for ONE K=4 SISO: 353k / (2*2) = **88k gates (logic)**
- No power estimates provided

[2] Chang, Suzuki, Parhi, “A 2-Mb/s 256 State 10-mW Rate-1/3 Viterbi Decoder,”

- A rival state-of- the art K=7, 256 State (Viterbi) convolutional decoder
- CMOS - 0.5um
- Area = 2.46 mm * 4.17 mm
- **Area scaled to 0.18um gates = 113.6k**
- Power @ 1.8 V, scaled to 10 Mbps = **50 mW**
- Note that this decoder does not iterate, divide by the number of iterations to compare
- **Power div (8 iterations * 2 decoders) = 3.125 mW**

  Therefore, for slightly less power (2.51 vs. 3.13 mW) than a standard Viterbi decode, I can perform an 8 iteration Turbo decode - with a total latency of 8 * 64 = 512 bits + interleaver, and get near Shannon error performance.

Clearly, the design presented here has the others significantly outdone in terms of area and power. It can also operate at speeds much higher than the rivals in its class.

6.0 References

[1] 3GPP, “Technical Specifications Group Radio Access Network: Muliplexing and channel coding (FDD)”, 3GPP TS 25.212 V3.2.0, March 2000(Release 1999).

[2] Chang, Suzuki, Parhi, “A 2-Mb/s 256 State 10-mW Rate-1/3 Viterbi Decoder,” IEEE Solid-State Cct., Vol 35. No 6. June 2000.

[3] C. Berrou, A. Glavieux, and P. Thitmajshima, “Near Shannon limit error-correcting coding and decoding. Turbo codes,” in Proc. Int. Conf. Communications, p. 1064-1070, May 1993

[4] Woodward, Jason and Hanzo, Lajos, “Comparative Study of Turbo Decoding Techniques: An Overview,” IEEE Trans. Vehicular Tech. Vol. 49, No. 6, November 2000.

[5] S.Benedetto, D. Divsalar, G. Montorsi and F. Pollara, “Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes,” TDA Progress Report 42-124. February 1996.

[6] Masera, Piccinini, Roch, Zamboni, “VLSI Architectures for Turbo Codes.” IEEE Trans. VLSI, Vol. 7, No. 3, September 1999

[7] - Viglione, Masera, Piccinini, Roch, Zamboni, “A 50 Mbit/s Iterative Turbo-Decoder.” Automation and Test in Europe Conference and Exhibition 2000. Proceedings, 2000

[8] Suzuki, Wang, Parhi, “A K=3 2 Mbps Low Power Turbo Decoder for 3rd Generation W-CDMA Systems.” IEEE Solid-State, pg 826 -834 Vol.35 No. 6, June 2000

[9] Advanced Hardware Architectures Inc, “Product Specification AHA4501 Astro 36 Mbits/sec Turbo Product Code Encoder/Decoder, 3.3V.” AHA, 2365 NE Hopkins Court, Pullman, Wa., Undated.

[10] Garrett, Stan, “A 2.5 Mb/s, 23 mW SOVA traceback chip for turbo decoding applications,” IEEE Symposium on Circuits and Systems, 2001. p 61-64 Vol. 4, ISCAS 2001

[11] Bickerstaff, Garrett, Prokop, Thomas, Widdup, Zhou, Nicol, Yan, “A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18 um CMOS,” Digest of Technical Papers, IEEE Solid State Circuits Conference, p.124 Vol. 1 Feb 2002