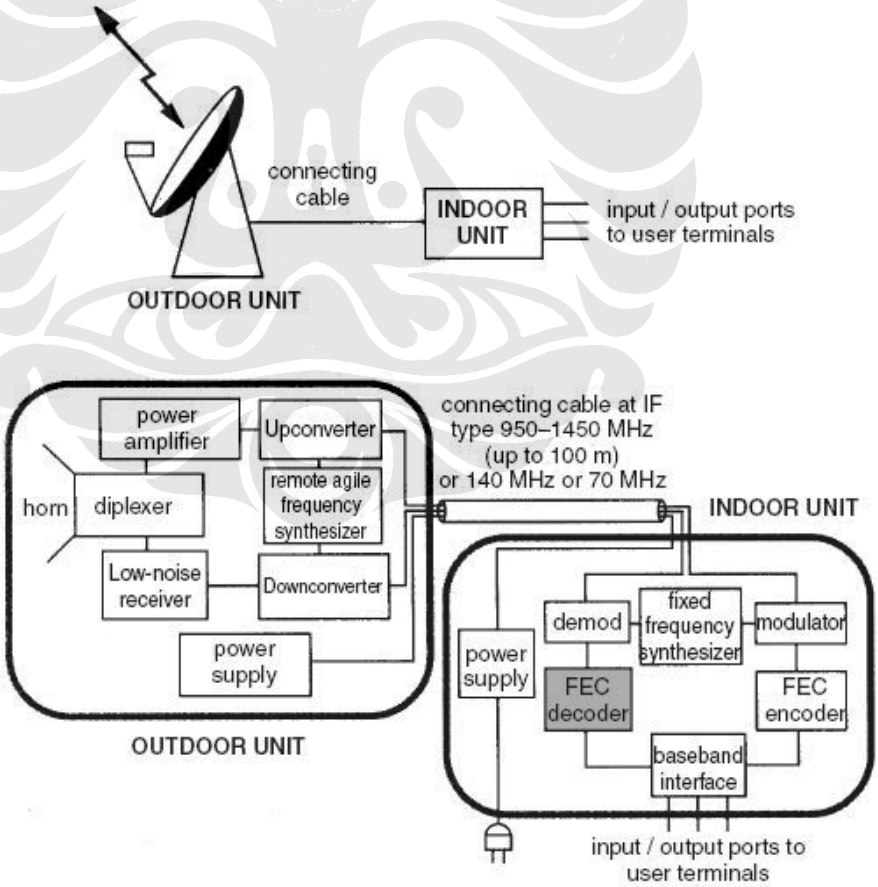
BAB 3

MEKANISME PENGKODEAAN *CONCATENATED VITERBI/REED-SOLOMON* DAN *TURBO*

Untuk proteksi terhadap kesalahan dalam transmisi, pada sinyal digital ditambahkan *bit – bit redundant* untuk mendeteksi kesalahan. Semakin banyak bit yang ditambahkan maka kemampuan untuk mendeteksi kesalahan dalam satu *frame* pengiriman akan lebih besar, dengan demikian *error* menjadi lebih kecil (*BER* lebih baik). Teknik penambahan *bit* tersebut disebut *error correction coding*.

Pada jaringan *VSAT* proses *error correction* terjadi pada blok *decoder* seperti yang ditunjukkan pada Gambar 3.1.

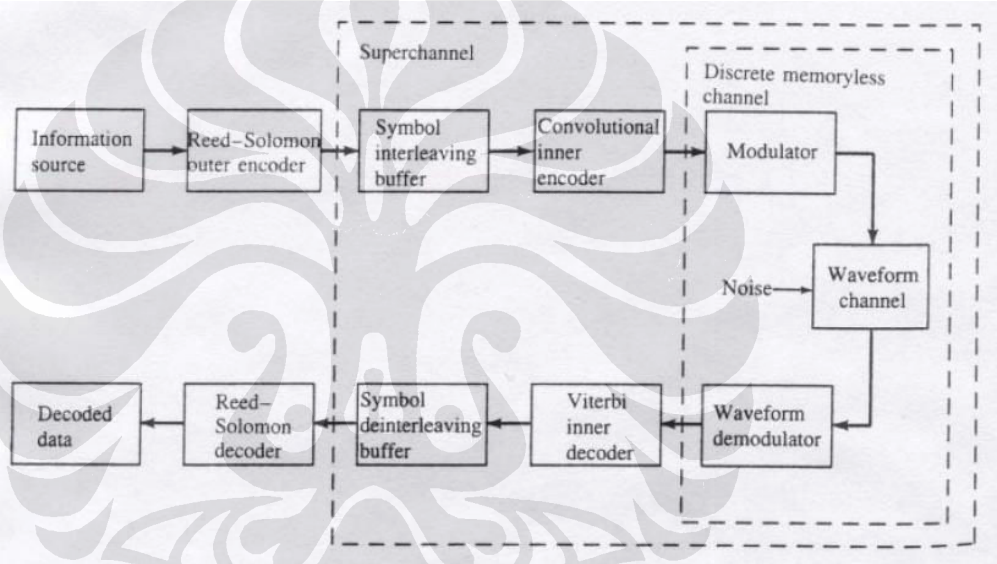


Gambar 3.1. *VSAT station equipment* [2] .

Pada tugas akhir ini dilakukan pengamatan sistem transmisi jaringan *VSAT* dengan menggunakan pengkodean *concatenated viterbi/reed-solomon* dan *turbo*. Pengamatan dilakukan pada sisi *decoder* untuk menganalisis performansi *BER*.

3.1 Pengkodean *Concatenated Viterbi/Reed-Solomon*

Untuk mengatasi adanya *burst error* yang terjadi pada *viterbi decoding*, maka teknik *concatenated* sangat diperlukan dengan menempatkan *convolutional encoder/viterbi decoder* sebagai *inner code* dan *reed-solomon* sebagai *outer code*. Digunkannya *reed-solomon* sebagai *outer code* dikarenakan pengkodean tersebut mampu mengatasi terjadinya *burst error*.



Gambar 3.2. *Serial concatenated viterbi* dengan *reed-solomon* [8].

Pada *serial concatenated code* blok *message* pertama kali dikodekan oleh *reed-solomon encoder* yang disebut *outer code*, pengkodean yang kedua dilakukan oleh *convolutional encoder* yang disebut *inner code*. *Decoding* pada *concatenated code* dilakukan dua tahap. Yang pertama oleh *viterbi decoding* yang disebut *inner code* dan kemudian oleh *reed-solomon* yang disebut *outer code*. Penggunaan *interleaver/deinterleaver* dimaksudkan untuk mendapatkan keluaran *high weight codeword.*

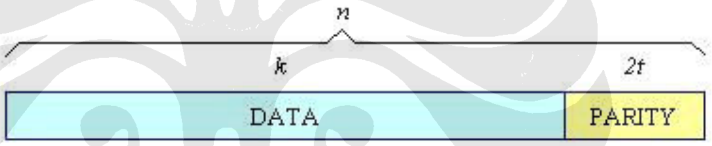
3.1.1 Pengkodean *Reed-Solomon (RS)*

Pengkodean *reed-solomon* merupakan kelas dari *linier, non-binary*, *cyclic block codes*. Kelas ini adalah *subfamily* dari *family linier, non-binary, cyclic* BCH *codes* yang merupakan generalisasi dari *Galois field GF(q).*

3.1.1.1 Properti Pengkodean *Reed-Solomon (RS)*

Sebuah *RS code* $C_{RS}(n,k)$ mampu untuk mengkoreksi *error pattern* dari ukuran t atau kurang yang didefinisikan melalui *Galois field GF(q).* Adapun parameternya :

Code length	$n = q - 1$
Number of parity check elements	$n - k = 2t$
Minimum distance	$d_{min} = 2t + 1$
Error-correction capability	t *element error per code vector*



Gambar 3.3. Diagram *reed-solomon* [9].

Jika α adalah *primitive element* dari *GF(q)* dan $\alpha^{q-1} = 1$, sebuah *RS code* $C_{RS}(n,k)$ dengan panjang n = q – 1 dan dimensi k adalah linier, *cyclic*, maka blok *RS code* dalam bentuk *polynomial* [9]:

$$\begin{aligned} g(X) &= (X - \alpha)(X - \alpha^2) \dots (X - \alpha^{n-k}) \\ &= (X - \alpha)(X - \alpha^2) \dots (X - \alpha^{2t}) \\ &= g_0 + g_1X + g_2X^2 + \dots + g_{2t-1}X^{2t-1} + g_{2t}X^{2t} \quad \text{.....(3.1)} \end{aligned}$$

3.1.1.2 Bentuk Sistematik *RS code*

Polynomial dibentuk dengan koefisien – koefisien yang merupakan unsur – unsur dari *Galois field GF(2^m).* Kode *polynomial* adalah perkalian dari generator *polynomial* g(X) termasuk semua akar – akarnya.

Bentuk *message polynomial* [9] :

$$m(X) = m_0 + m_1X + \dots + m_{k-1}X^{k-1} \quad \text{.....(3.2)}$$

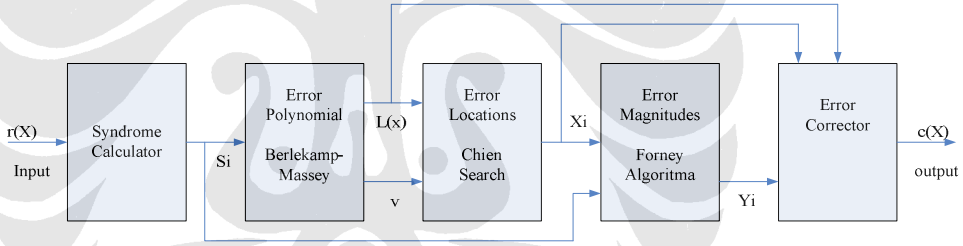
Message polynomial ini juga dibentuk dengan koefisien – koefisien yang merupakan elemen – elemen dari *Galois field GF(2^m)*. Bentuk sistematik dari pengkodean ini diperoleh dengan cara yang sama untuk *binary BCH codes*.

Adapun bentuk sistematik *RS code* [9]:

$$X^{n-k}m(X) = q(X)g(X) + p(X) \quad \text{.....(3.3)}$$

3.1.1.3 *Decoder Reed-Solomon*

Reed-solomon decoder mencoba untuk mengkoreksi *error* dengan menghitung *syndromes* untuk setiap *codeword*. *Syndromes decoder* mampu menentukan jumlah *error* pada blok penerima. Jika ada *error* yang terjadi, *decoder* mencoba untuk menemukan lokasi *error* menggunakan algoritma *Berlekamp-Massey*. Untuk menemukan akar – akar *polynomial* menggunakan algoritma *Chien search*. Algoritma *Forney's* digunakan untuk menemukan nilai simbol *error* dan mengkoreksinya. Untuk sebuah *RS* (n , k) kode di mana n – k = 2t, *decoder* dapat mengkoreksi sampai t simbol *error* pada *codeword*.



Gambar 3.4. Arsitektur umum *decoder RS* [9].

Di mana :

r(x) : *Received codeword*

Si : *Syndromes*

L(x) : *Error locator polynomial*

Xi : *Error locations*

Yi : *Error magnitudes*

c(x) : *Recovered codeword*

v : *number of error*

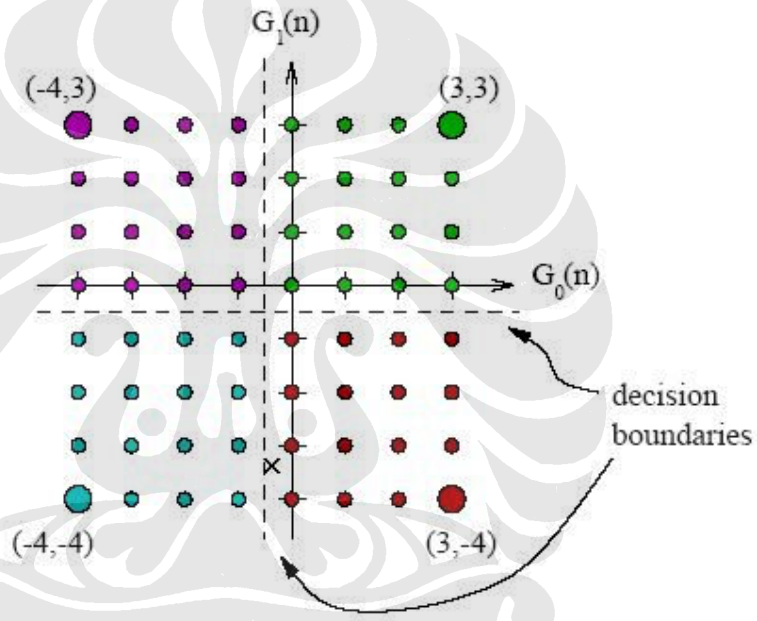
3.1.2 Pengkodean *Viterbi*

Algoritma *viterbi* adalah metode yang umumnya digunakan untuk medekodekan *bit streams* yang dikodekan oleh *convolution coders*. Algoritma

viterbi bukan satu – satunya algoritma yang bisa digunakan untuk mendekodekan *bit streams* dari *convolution coder*.

3.1.2.1 *Viterbi Decoding*

Viterbi decoding dapat rinci menjadi dua operasi yaitu *metric update* dan *traceback*. Pada *metric update*, dua hal dikerjakan pada setiap simbol interval (1 simbol = 1 *input* bit = 2 *encoded bits*). Akumulasi *state metric* di kalkulasi untuk setiap *state* dan optimal *incoming path* dihubungkan dengan setiap *state* yang ditentukan. *Traceback* menggunakan informasi ini untuk memperoleh optimal *path* melalui *trellis*.

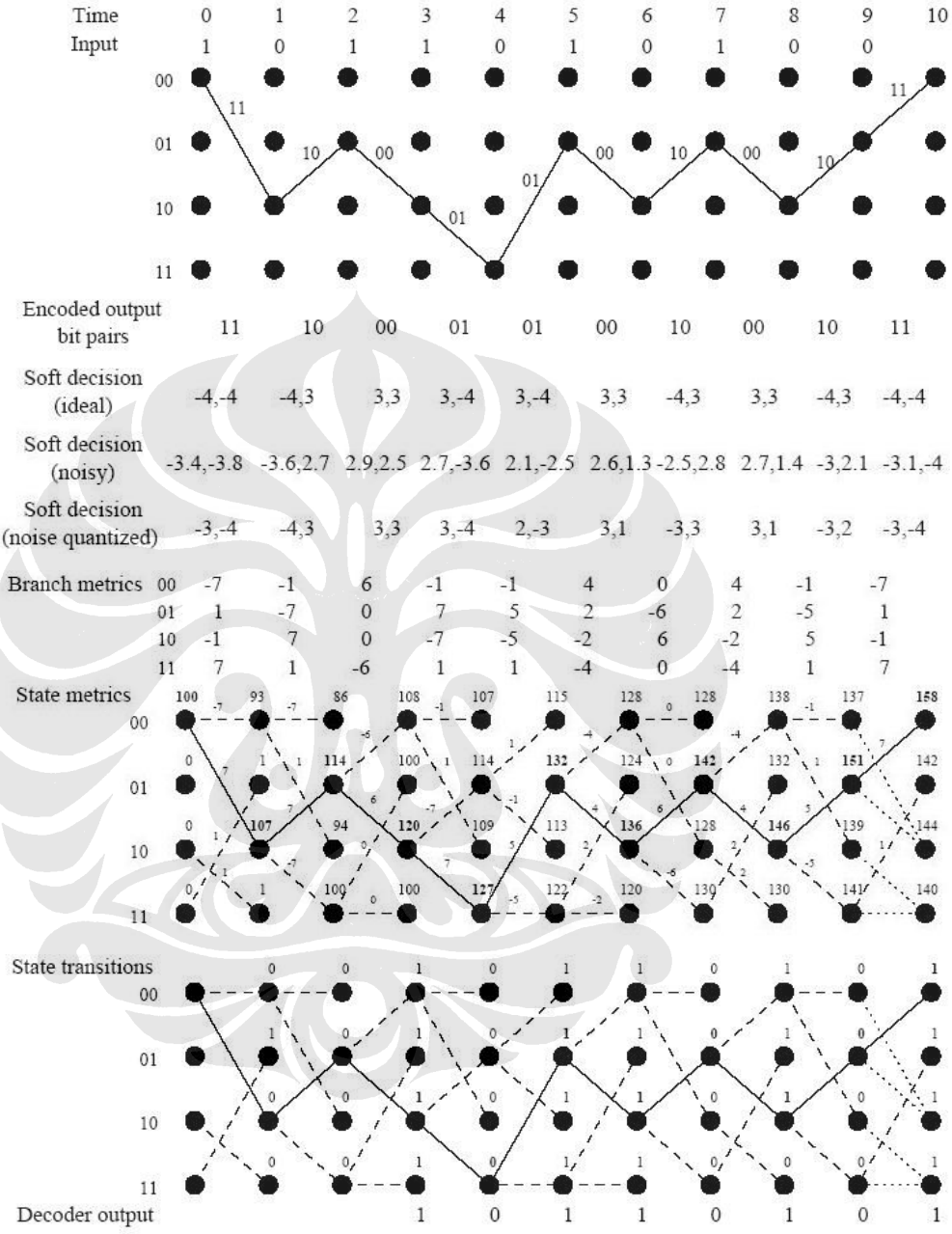


Gambar 3.5. Sinyal *constellation* untuk diteksi *symbol by symbol* [10].

Untuk memahami operasi *metric* yang digunakan pada viterbi *decoding*, pasangan bit ($G_0(n)$, $G_1(n)$) dianggap sebagai *encoded* bit yang diterima. Misalkan 1 1 ditransmisikan, kita mengharapkan 1 1 diterima dengan *soft decision pair* dari (-4, -4). Akan tetapi , *noise channel* mungkin merusak sinyal yang ditransmisikan sehingga *soft decision pair* yang diterima (-0.4, -3.3) dan nilai hasil kuantisasi (0, -3) yang dapat dilihat pada Gambar 3.5.

Traceback dimulai setelah *metric update* menyelesaikan simbol terakhir pada *frame*. Transisi *state* dibutuhkan oleh *traceback* algoritma untuk *frame by frame* viterbi *decoding*.

Contoh *convolutional encoding* dan *viterbi decoding* dengan data input 1011010100 [10] :

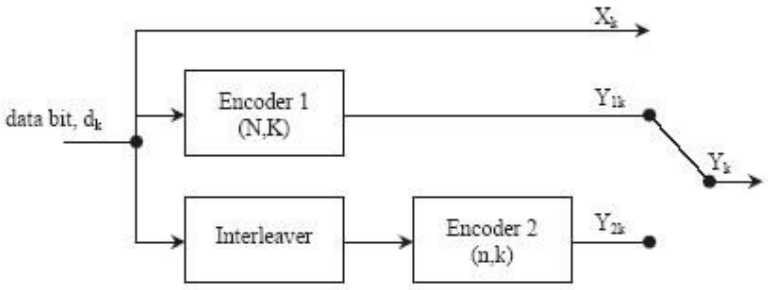


3.2 Pengkodean *Turbo*

3.2.1 *Parallel Concatenation*

Pengkodean *turbo* adalah *parallel concatenation* dari dua atau lebih sistematik pengkodean. Sistematik pengkodean adalah salah satu dari keluaran

menjadi bit – bit masukan. Gambar 3.6 menunjukkan blok diagram dari *rate* 1/3 turbo *encoder* [11].



Gambar 3.6. *Parallel concatenation* [11].

Keluaran dari *encoder* pada waktu k adalah X_k dan Y_k. X_k selalu sama dengan d_k masukan dari data bit dan Y_k terdiri dari Y_{1k} dan Y_{2k} yang merupakan keluaran dari *encoder* 1 dan *encoder* 2. Ketika keluaran dari *encoder* 1 dipilih pada waktu n_1 dan keluaran dari *encoder* 2 dipilih pada waktu n_2, maka ratenya adalah [11]:

$$R_1 = \frac{n_1+n_2}{2n_1 \mid n_2} \quad \text{dan} \quad R_2 = \frac{n_1+n_2}{2n_2 \mid n_1} \qquad \text{.....(3.4)}$$

Adapun *rate R* dari hasil pengkodean [11]:

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1 \qquad \text{.....(3.5)}$$

Satu keuntungan yang penting menggunakan *parallel concatenation* adalah diizinkan menggunakan *single clock feeding* pada kedua *encoder*.

3.2.2 *Interleaver*

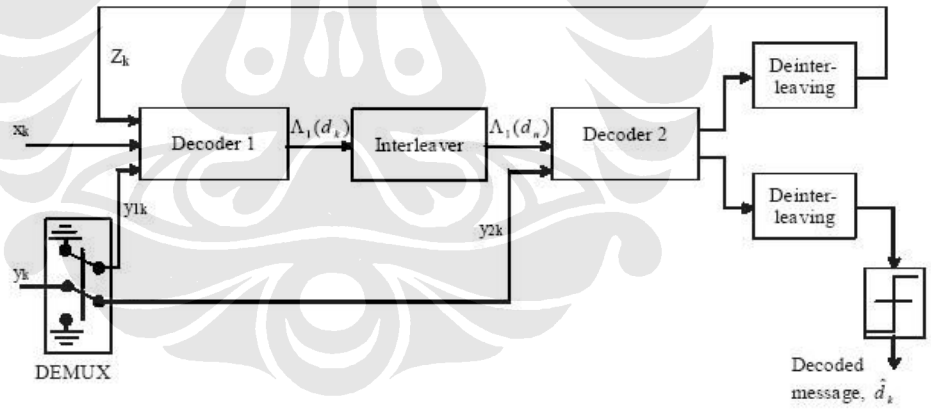
Interleaver diletakkan sebelum masukan *bit sequence* dilalui oleh *encoder* 2, seperti yang ditunjukkan pada Gambar 3.6. Adapun tujuan meletakkan *interleaver* sebelum *encoder* 2 adalah untuk mendapatkan keluaran *high weight codeword* pada *encoder* 2 jika keluaran dari *encoder* 1 menjadi *low weight codeword*. Penggunaan *interleaver* pada pengkodean *turbo* adalah untuk mengurangi *bit – bit error* dengan mendapatkan keluaran *high weight codeword* dari *encoder*.

3.2.3 *Puncturing*

Puncturing didefinisikan sebagai penghapusan selektif dari beberapa *parity bits* [12]. *Puncturing* digunakan untuk menigkatkan *coded data rate*. Mengacu pada Gambar 3.6, *puncturing* dapat tercapai dengan membawa *bit – bit* genap ke *encoder* 1 dan *bit – bit* ganjil ke *encoder* 2. Adapun *bit – bit* ganjil pada *encoder* 1 dan *bit – bit* genap pada *encoder* 2 dihapus. Pada kasus ini, *coded bit rate* menigkat dari 1/3 menjadi 1/2. Operasi *puncturing* dilakukan dengan menggunakan *multiplexer*.

3.2.4 *Turbo Decoding*

Turbo decoder terdiri dari dua *decoder* dasar yang disusun secara *serial*, seperti yang ditunjukkan pada Gambar 3.7. Masukan *decoder* 1 adalah sistematik bit X_k, informasi *redundant encoding* y_{1k} dan yang ke tiga $Z_{k,}$ yang merupakan *feedback* dari *decoder* 2. *Decoder* 2 menerima informasi dari *decoder* 1 dan masukan *redundant* y_{2k}. Kemudian *decoder* 2 membuat keputusan terakhir berdasarkan informasi kumulatif dari setiap *bit*.



Gambar 3.7. Turbo decoder [11].

Persyaratan yang penting untuk *iterative decoding* adalah kemampuan untuk mengkalkulasi *soft decision values* untuk setiap data *bit*. Oleh karena itu, dipilih pendekatan yang memproses *bit – bit* oleh *decoding* algoritma yang memaksimalkan kemungkinan *posteriori* dari setiap *bit*. Algoritma itu disebut

Maximum A posteriori Probability (MAP) . Statistik *soft decision* umumnya diwakili oleh *Log Likelihood Ratio (LLR)* .

Algoritma *MAP* menggunakan *LLR* sebagai sebuah *metric*. *LLR* tidak segera tersedia dan dihitung oleh terminologi sederhana.

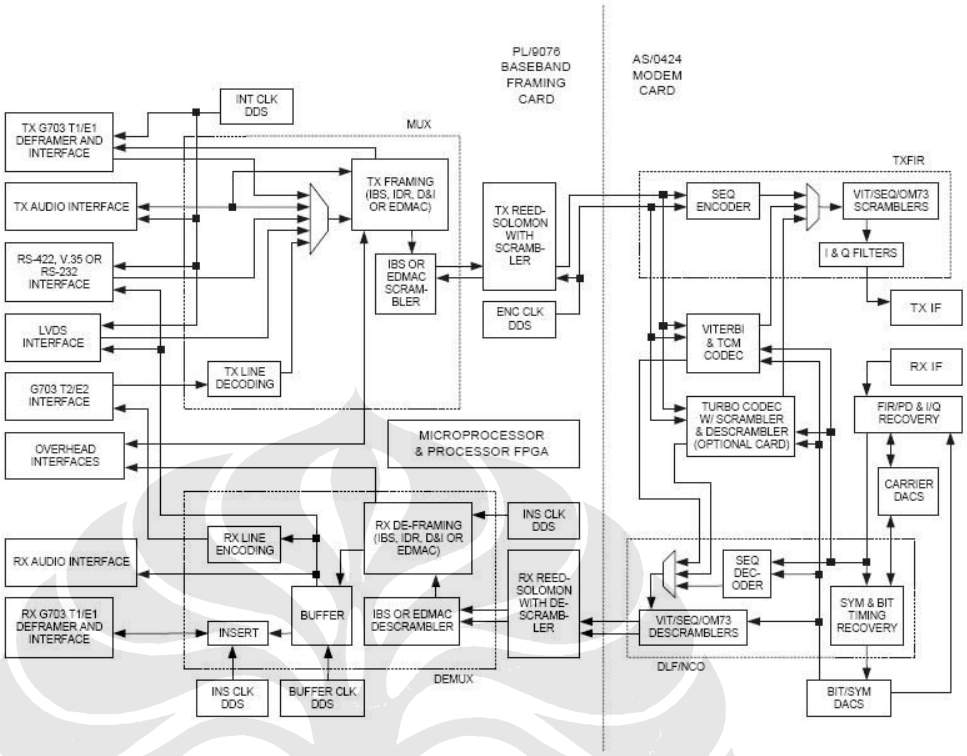
Algoritma *MAP* memisahkan *LLR* ke dalam 3 *terms* yaitu α, β dan γ yang memungkinkan untuk menghitung pada observasi dari seluruh urutan yang diterima

3.3 Blok Diagram *Modem* Comtech CDM 600

Modem Comtech CDM 600 mempunyai dua tipe *interface* yang berbeda yaitu *IF* dan data. *Interface* data menghubungkan perangkat pelanggan dan *modem*. Sedangkan *interface IF* menghubungkan *modem* ke satelit melalui perangkat *uplink* dan *downlink*.

Transmit data diterima oleh *terrestrial interface* dimana jalur penerima mengkonversi sinyal – sinyal *clock* dan data menjadi *CMOS* level untuk proses selanjutnya. Jika *framing* diaktifkan, transmit *clock* dan keluaran data dari *FIFO (First In First Out)* akan dilewatkan ke dalam bentuk *frame* yang diinginkan. Jika *framing* tidak diaktifkan *clock* dan data akan dilewatkan secara langsung ke *forward Error Correction encoder*. Di *FEC encoder* data dikodekan sesuai dengan pengkodean yang digunakan. Melalui *encoder*, data di*filter* oleh transmit *digital filter* yang menunjukkan bentuk *spectral* dari sinyal – sinyal data. *Resultant* dari sinyal *I* dan *Q* dimasukkan ke dalam *BPSK, QPSK, 8-PSK* atau *16-QAM modulator. Carrier* dihasilkan *oleh frequency synthesizer* dan sinyal - sinyal *I* dan *Q* memodulasi *carrier* untuk menghasilkan keluaran sinyal *IF*.

Pada sisi penerima, sinyal *Rx –IF* diterjemahkan menjadi sinya *baseband* menggunakan *carrier recovery VCO*. Hasil dari sinyal tersebut dipisah menjadi komponen *in-phase (I)* dan *quadrature (Q).* Kemudian sinyal *I* dan *Q* di lakukan *sampling* oleh *A/D converter*. Hasil demodulasi sinyal dimasukkan ke dalam *FEC decoder* (viterbi, reed-solomon atau turbo). Setelah *decoding* dilakukan, sinyal *clock* dan data dilakukan *de-framing* jika *framing* diaktifkan. Jika tidak data akan dilalui ke *Doppler buffer*. Sinyal *clock* dan data yang diterima akan diarahkan ke *terrestrial interface.*



Gambar 3.8. Blok Diagram *Modem* CDM 600 [13].