X. PRAKTIKUM 9— Algoritma Sekuensial Fano untuk Dekode FEC Konvolusi

X.1 PENDAHULUAN

Paket data WSPR menggunakan bit paritas yang disebut forward error correction (FEC) menggunakan skema kode konvolusional. Proses decoding dilakukan oleh dekode sekuensial di mana memiliki dua opsi yang disebut algoritma Fano sebagai default, dan opsi lain adalah algoritma Jalinek. Bagian dari perangkat lunak WSPR ini dikembangkan dan disumbangkan oleh P. Karn, seorang engineer senior di pabrik IC QUALCOMM milik Viterbi, dan, dan selanjutnya baru-baru ini, terintegrasi dengan paket WSPR menjadi perangkat lunak open source. Perangkat lunak ini telah ditulis dengan sangat elegan oleh kontributor, sehingga, dengan melacak dan mengamati sintaksis, para mahasiswa diharapkan dapat dengan mudah memahami tujuan dari masing-masing baris sintaksis dalam program.

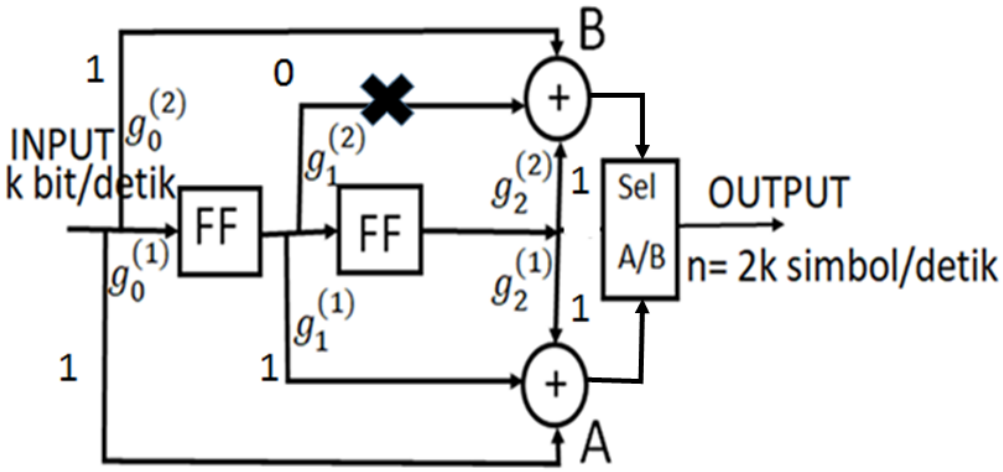
Beberapa perihal penting dimana para mahasiswa dapat membaca manual percobaan sambil melacak dan mengamati sintaksis program adalah sebagai berikut:

- Menurunkan rumus fungsi kerapatan probabilitas normal untuk secara praktis dapat dipakai menghitung probabilitas transisi yang terjadi pada simbol (tegangan) yang diperoleh pada terminal sound-card, saat digit “0”atau “1” dikirim dari sisi pemancar. Hal ini juga sudah dibahas pada PRAKTIKUM 8.
- Membangkitkan tabel metrik untuk soft-decision dekoder konvolusi. Hal ini sudah dibahas pada PRAKTIKUM 8.
- Selanjutnya Praktikum ini kita akan membahas implementasi decoder konvolusi soft-decision dengan menggunakan algoritma Fano, termasuk diantaranya adalah operasi bitwise dengan menggunakan bahasa C saat melaksanakan proses decoding, deinterleaving hingga unpacking data.

X.2 Algoritma Viterbi

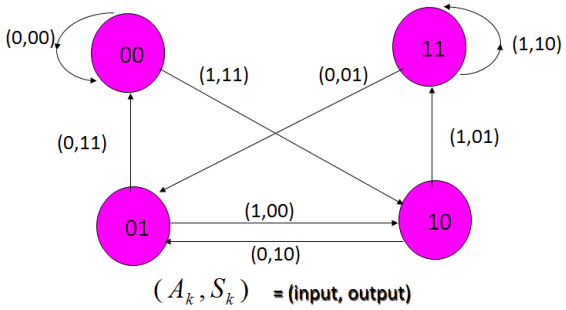
Sebelum malaksanakan percobaan algoritma Fano, akan kita bahas terlebih dahulu algoritma Viterbi yang lebih sederhana dengan delay hanya 3. Contoh kode dan dekode konvolusi untuk kanal bernois yang akan kita bahas secara sepintas mempunyai spesifikasi sbb,

Panjang kodeword 2, panjang bit data/message/berita adalah 1 (artinya 1 bit berita akan dikodekan menjadi 2 bit kode), dan panjang delay konvolusi adalah 3, sehingga sering disingkat menjadi (n,k,v) = (2,1,3), Jika ditentukan kode generator 7 dan 5 atau 111 dan 101 maka, diagram konvolusi akan mempunyai bentuk seperti pada Gambar X-1 dibawah ini,



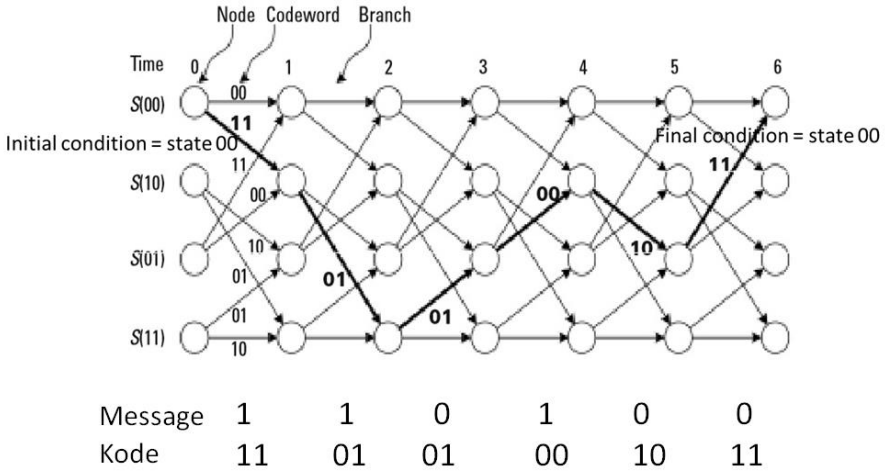
Gambar X-1: Blok diagram dengan kode konvolusi (2,1,3) dengan generator kode 7 (111) dan 5 (101).

Selanjutnya dari blok diagram tersebut dapat dibuat diagram state seperti pada Gambar X-2 dibawah ini,



Gambar X-2: Diagram state dari kode konvolusi (2,1,3)

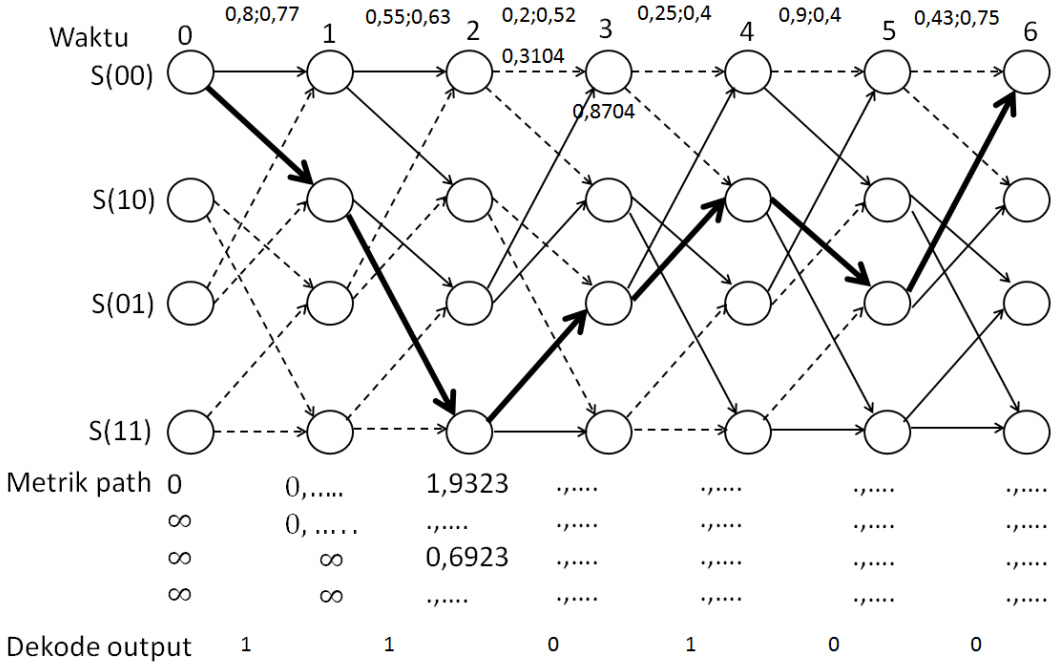
Selanjutnya jika mengirim data (1 1 0 1 0 0), maka diagram trellis kode konvolusi mempunyai bentuk seperti pada gambar dibawah ini,



Gambar X-3: Diagram trellis kode konvolusi (2,1,3) dengan kode generator 7 (111) dan 5 (101)

Dapat dilihat pada Gambar X-3, bahwa setiap node terdapat kemungkinan 4 state. Ini artinya dengan delay v=3 maka akan ada $2^{v-1} = 2^{3-1} = 4$ kemungkinan state. Trellis tersebut mengonversi message biner (110100) menjadi kode (word) biner (110101001011).

Selanjutnya disisi penerima akan berusaha untuk mereplika jalur trellis ini agar dapat mendekode deretan data (terganggu nois) yang diterima disisi receiver (penerima). Salah satu cara mereplika jalur trellis adalah metoda Viterbi yang tampak pada Gambar X-4.



Gambar X-4: Algoritma Viterbi untuk mereplika path trellis yang dibangkitkan dari sisi pemancar.

Algoritma Viterbi diadopsi untuk mereplika path yang telah dibangkitkan dari sisi pemancar. Untuk memeriksa path diperlukan perhitungan jarak antar node sebanyak $2^{v-1} \times L$, dimana L adalah panjang sekuen. Dapat dilihat bahwa beban komputasi dari algoritma Viterbi akan naik secara linear dengan bertambahnya panjang sekuen L.

Dua metrik yang dipergunakan dalam algoritma Viterbi yakni, metrik-branch **(MB)** dan metrik-path **(MP).** Jika diketahui $c=(c^{(1)}\ c^{(2)}...\ c^{(n)})$ adalah codeword yang dikirim oleh pemancar, dan $r=(r^{(1)}\ r^{(2)}...\ r^{(n)})$ adalah deretan vektor/simbol yang diterima di sisi penerima. Selanjutnya didefinisikan bahwa suatu metrik-branch **(MB)** adalah,

$$MB_{(r,c)} = \begin{cases} d_H(r,c), & BSC \\ \sum_{i=1}^{n} \left| r^{(i)} - c^{(i)} \right|^2, & kanal\ AWGN \end{cases}$$

Dimana **BSC** adalah binary Symateric Channel, dan r dapat berupa kode biner atau bilangan yang bernilai melambangkan kode biner. Metrik-branch adalah ukuran seberapa mirip simbol yang diterima dengan codeword yang dikirim pemancar.

Metrik yang lain adalah metrik-path MP dari state S, yang merupakan jarak akumulasi dari metrik-branch MB yang dihitung dari awal trellis sampai ke titik dekoding sekarang, yakni,

$$MP_{(S,t)} = \sum_{\{branch\}} MB_{(r,c)}$$

Dimana {branch} melambangkan semua branch yang membentuk path. Selanjutnya metrik-path dapat dihitung secara rekursif seperti dibawah ini,

$$MP_{(S,t+1)} = MP_{(S',t)} + MB_{(S',t)\rightarrow(S,t+1)}$$

Artinya adalah bahwa metrik-path dari node depan (S,t+1) adalah metrik-path dari node sekarang (S',t) ditambah dengan metrik-branch node (S',t) ke node (S,t+1) atau dengan kalimat matematika ditulis (S',t)→(S,t+1).

Selanjutnya perhatikan contoh berikut ini,

Jika dikirimkan suatu sekuen kode $\vec{c} = 110101001011$ (proses pengodean dari $\vec{m} = 110100$). Setelah melalui kanal bernois, pada sisi receiver diterima sekuen simbol vektor

$$\vec{r} = 0{,}8\ 0{,}77\ 0{,}55\ 0{,}63\ 0{,}2\ 0{,}52\ 0{,}25\ 0{,}4\ 0{,}9\ 0{,}4\ 0{,}43\ 0{,}75$$

Algoritma Viterbi terbagi menjadi 3 prosedur:

1. Menghitung semua $MB_{(r,c)}$ dengan rumus seperti diatas, dan hasilnya seperti ditunjukkan seperti pada gambar 1-4. Sebagai contoh untuk branch S(00)→S(00) pada t=3, dimana simbol vektor yang diterima adalah (0,2 0,52), MB dihitung dengan cara $|0 - 0{,}2|^2 + |0 - 0{,}52|^2 = 0{,}3104$. Perhatikan, bahwa biner 0 0 bukan angka biner dari state S(00),

melainkan adalah proses koding dari S(00) menuju S(00) saat input digit biner 0 menghasilkan biner 00, dan selanjutnya biner

2. [0 0] inilah yang diukur jaraknya terhadap simbol [0,2 0,52].
3. Selanjutnya adalah menghitung semua metrik-path.
4. Kemudian path dengan nilai MP paling kecil diambil dan path yang lain dibuang. Sebagai contoh lihat (S(00),3), terdapat 2 path menuju node ini yakni, node (S(00),2) dan node (S(01),2). Metrik path menuju (S(00),3) adalah:
   1,9323 [metrik path dari node (S(00),2)] + 0,3104 (metrik branch) = 2,2427, dan
   0,6923 [metrik path dari node (S(01), 2)] + 0,8704 (metrik branch) = 1,5627. Selanjutnya metrik dengan nilai 1,5627 akan dipilih sebagai "path surviver" yang akan dipertahankan, lihat garis tegas pada gambar 1-4, sedangkan path yang lain (garis putus-putus) akan dibuang. Sesuai "perjanjian" bahwa enkoder berawal dari node (S(00),0) maka algoritma Viterbi ini juga berawal dari (S(00),0). Jika proses berakhir pada node (S(00),6), dan selanjutnya sudah terpilih global optimum path yakni S(00)→S(10)→ S(11)→S(01)→ S(10)→S(01)→ S(00), yang ditunjukkan dengan garis tebal. Selanjutnya berdasarkan diagram state, global path dikonversi menjadi deretan digit biner hasil proses dekode, sbb, S(00)→S(10) menghasilkan digit 1, S(10)→S(11) menghasilkan digit 1, S(11)→S(01) meghasilkan digit 0, dan seterusnya sehingga secara keseluruhan hasil dekode adalah 1 1 0 1 0 0 yang merupakan replika dari sekuen digit yang dikirim oleh pemancar.

   Proses diatas lazim disebut dekode soft-decision karena simbol yang diterima merupakan nilai besaran fisik/tegangan-listrik yang sebenarnya. Lain halnya jika asumsi kanal komunikasi adalah kanal BSC dan digit yang diterima adalah sekuen biner, maka proses dekode dikatakan sebagai dekode hard-decision.

   Pemilihan path untuk mendapat path-optimum-global mengandalkan operasi ADD, COMPARE (untuk mencari path dengan metrik minimum) dan SELECT (untuk memilih path dengan metrik minimum yang kemudian ditetapkan sebagai "path survivor"). Operasi ini kemudian biasa disebut sebagai operasi ACS (add,compare dan select).

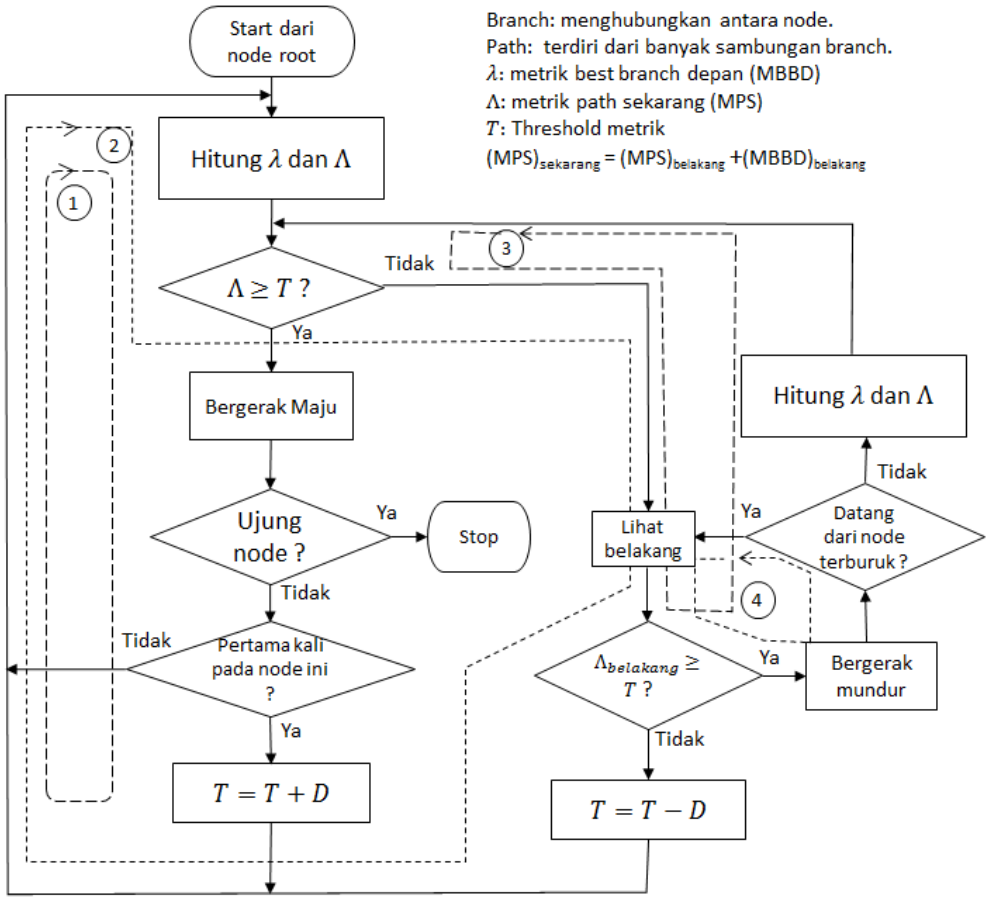
X.3 Algoritma Fano

Algoritma dekode Viterbi mengandalkan sediaan matriks $2^{v-1} \times L$ dengan strutur fix (tetap) sebagai platform proses pemrograman dinamik (trellis). Untuk itulah maka agoritma Viterbi hanya efisien untuk jumlah delai (v) maupun panjang sekuen yang kecil. Algoritma Viterbi mempunyai waktu pemrosesan yang fix/tetap dan terprediksi. Untuk itulah maka algoritma Viterbi banyak dipakai untuk aplikasi pengiriman data waktu nyata misalnya sistem teleponi digital. Sedangkan untuk jumlah v yang besar dan sekuen data relatif panjang maka algoritma Viterbi yang bekerja secara paralel ini menjadi tidak dapat diterapkan karena perlu memori besar, maka diperlukan suatu

algoritma sekuensial untuk dapat menyelesaikan pemrograman dinamik dengan nilai v dan L yang besar.

Untuk melaksanakan dekode kode konvolusi dikenal 2 kategori, yakni yang mengandalkan algoritma paralel yang biasa disebut algoritma Viterbi, yang yang mengandalkan algoritma sekuensial yang salah satu yang terkenal adalah algoritma Fano. Karena dilaksanakan secara paralel, maka algoritma Viterbi mempunyai waktu prosesing yang lebih pasti, selain dari pada itu algoritma Viterbi lebih cocok untuk implementasi chipset dan hanya cocok untuk algoritma konvolusi dengan faktor tunda atau angka konstrain (***v***) lebih kecil dari 10. Dalam proses pencarian path alogoritma dekode konvolusi, istilah andalan yang dipakai adalah "better path", yang menggambarkan bahwa suatu path mempunyai kemungkinan lebih benar dibanding path yang lain yang sedang dibandingkan. Dekode sekuensial mengandalkan pendekatan perbandingan "mana lebih baik" sehingga secara komputasi lebih meringankan dan masuk akal untuk diiplementasikan. Meskipun dikatakan bahwa kinerja dekode sekuensial tidak sebaik dekode paralel, namun untuk angka konvolusi mendekati 50, maka tidak ada pilihan bahwa dekode sekuensial adalah menarik untuk diimplementasikan karena dekode sekuensial mempunyai kompleksitas yang tak tegantung pada panjang konstrain/delay ***(v)***.

Algoritma sekuensial biasa dipakai untuk menyelesaikan pemrograman dinamik dengan v dan L besar. Algoritma sekuensial mempunyai kekurangan yakni titak mempunyai waktu pemrosesan yang tetap, yaitu tergantung dari nilai SNR, namun karena mempunyai delai besar maka kinerja koreksi eror algoritma Fano sangat lebih baik dibanding dengan sistem dengan nilai delai kecil. Semakin kecil nilai SNR maka semakin lama waktu prosesingnya. Algoritma Fano banyak dipakai untuk pengiriman data bukan waktu nyata lewat kanal komunikasi yang sangat bernois misalnya pengiriman data dari stasiun ruang angkasa jauh (deep space) misalnya dari wahana Voyager. Ada dua implementasi dari dekode sekuensial, yakni algoritma Fano dan algoritma stack yang dikembangkan oleh Jelenik. Pada program WSPR terdapat 2 pilihan algoritma yang Fano dan Jelenik. Untuk praktikum ini kita akan bahas algoritma Fano.



Gambar X-5: Algoritma sekuensial Fano

Gambar X-5 menunjukkan algoritma sekuensial Fano. Kunci pelaksanaan algoritma ini adalah perbandingan terus menerus metrik path ***Λ*** dengan harga threshold T yang selalu berubah nilai. Pada saat ***Λ*** melebihi nilai T maka dekoder menganggap bahwa path telah mempunyai jalur yang benar dan terus bergerak maju. Jika ***Λ*** mempunyai nilai jatuh dibawah T, maka algoritma akan kembali kebelakang untuk mencari path yang lebih baik (memiliki nilai lebih besar dari T). Pada saat itu nilai T juga dikurangi dengan suatu harga Δ dengan maksud untuk melonggarkan algoritma bergerak kedepan. Algoritma Fano mengandalkan 3 paramater penting yakni, metrik path sekarang MPS (Λ_C), metrik best-branch-depan MBBD (Λ_S) dan metrik path belakang (Λ_P).

Penjelasan dari diagram blok gambar X-5 adalah sbb,

1. Inisialisasi berawal dari root dari pohon kode dengan memasang metrik path ***Λ*** dan threshold T sama dengan nol.
2. Saat ini pada node C (current), algoritma berupaya untuk bergerak maju dengan melakukan upaya-upaya sbb: pertama-tama, menghitung metrik untuk semua cabang (branch) 2^kyang mungkin, dan dipilih cabang yang mempunyai metrik paling besar, selanjutnya dipilih sebagai path yang paling baik (best path) saat sekarang. Kemudian Λ_S dihitung dengan formulasi dibawah ini,

$$\Lambda_S = \Lambda_C + \lambda_{C \to S}$$

Dimana,

Λ_S : metrik path-depan

Λ_C : metrik path-sekarang

$\lambda_{C \to S}$: metrik branch depan terbaik/terbesar

$C \to S$: transisi terbaik dari node sekarang ke node depan

Jika $\Lambda_S \geq T$ maka algoritma maju ke node S (successor). Jika algoritma sampai pada node S untuk pertama kali maka pada titik ini harga T diketatkan menjadi T+Δ. Jika sebaliknya ternyata algoritma sudah pernah sampai ke node S ini, maka nilai T tidak berubah. Algoritma ini terus berulang hingga sampai pada ujung pohon dekode. Prosedur ini merupakan loop utama atau loop ① seperti tampak pada Gambar X-5.

3. Jika pada suatu titik didapat kondisi bahwa $\Lambda_S < T$ dan $\Lambda_P < T$ maka algoritma akan menurunkan/melonggarkan nilai T dengan T-Δ, dan selanjutnya algoritma tetap bergerak maju dengan menjalankan loop ① namun dengan harga T= T-Δ. Prosedur ini merupakan loop ② seperti tampak pada Gambar X-5.

4. Prosedur terakhir adalah dengan mempertimbangkan bahwa terdapat kasus $\Lambda_S < T$ dan $\Lambda_P \geq T$. Dalam kasus ini algoritma pertama-tama akan bergerak mundur ke node P (predecessor), kemudian mencoba untuk bergerak maju dengan cara mencari "best-branch-depan" dengan asumsi bahwa "best-branch-depan" bukanlah suatu "branch-terburuk". Jika "best-branch-depan" tidak ada lagi, karena best-branch-depan ini adalah "branch-terburuk", maka algoritma akan mundur selangkah lagi untuk selanjutnya mencari "best-branch-depan" dari node ini. Prosedur ini merupakan loop ③ atau loop ④ seperti tampak pada Gambar X-5.

X.7 Pembangkitan Tabel State dan Proses Baca Tabel Enkoder (2,1,32)

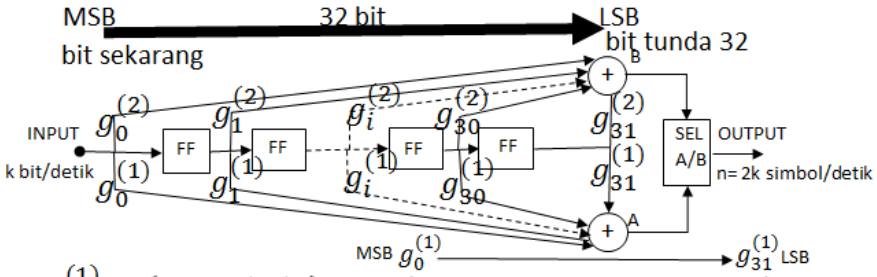
Tabel state selebar 32 bit tidak disimpan dalam array yang dinilai akan menghabiskan memori, namun dengan cara dibangkitkan dengan preprosesor direktif dalam bahasa C seperti pada sintaksis dibawah ini. Sintaksis preprosesor direktif ini diletakkan pada file **fano.h.**

```
#define         ENCODE(sym,encstate){\
unsigned long _tmp;\
\
_tmp = (encstate) & POLY1;\
_tmp ^= _tmp >> 16;\
(sym) = Partab[(_tmp ^ (_tmp >> 8)) & 0xff] << 1;\
_tmp = (encstate) & POLY2;\
_tmp ^= _tmp >> 16;\
(sym) |= Partab[(_tmp ^ (_tmp >> 8)) & 0xff];\
```

}

Dalam bahasa C, penulisan sintaksis preprosesor direktif selalu diawali dengan tanda “#” dan memerlukan tanda “\” untuk ganti baris dan enter yang bermakna eksekusi dari satu baris sintaksis. Preprosesor yang lazim dipakai adalah “#include” dan “#define”.

Preprosesor direktif ini membangkitkan kode konvolusi berdasarkan deretan 32 digit yang sedang mengantri untuk diberangkatkan. Proses ini melibatkan operasi bitwise (berbasis bit), sehingga perlu ditelaah dengan lebih seksama. Proses ini digambarkan pada blok diagram dibawah ini,



$$g_i^{(1)} = (F2D05351)_{16} = (1111\ 0010\ 1101\ 0000\ \ldots)_2$$

$$g_i^{(2)} = (E4613C47)_{16} = (1110\ 0100\ 0110\ 0001\ \ldots)_2$$

Gambar X-6: Enkoder Konvolusi (2,1,32)

Kode generator $g_i^{(1)}$ dan $g_i^{(2)}$ berfungsi sebagai saklar sebelum semua digit diumpankan pada elemen EXCLUSIVE-OR “atas” dan “bawah” sehingga menghasilkan 2 bit untuk setiap 1 bit input yang didorong dari sisi kiri ke kanan.

Proses EXCLUSIVE-OR ini sebenarnya hanyalah proses menghitung berapa banyak digit “1” dari 32 deretan bit. Jika jumlah bit “1” adalah ganjil maka menghasilan bit enkode “1”, demikian sebaliknya jika jumlah bit “1” adalah genab maka menghasilkan bit enkode “0”.

Namun demikian oleh developer program ini telah dikembangkan cara cepat untuk menghitung bit enkode, dengan melibatkan tabel Partab[i] yang telah dihitung terlebih dahulu dan disimpan pada file tab.c. Proses enkoder ini dapat diamati dengan mengeksekusi program **encode.c** pada langkah-langkah PERCOBAAN 9.

X.5 Memasang Symbol Hasil Penerimaan Radio Pada Array Untuk Proses Dekoding

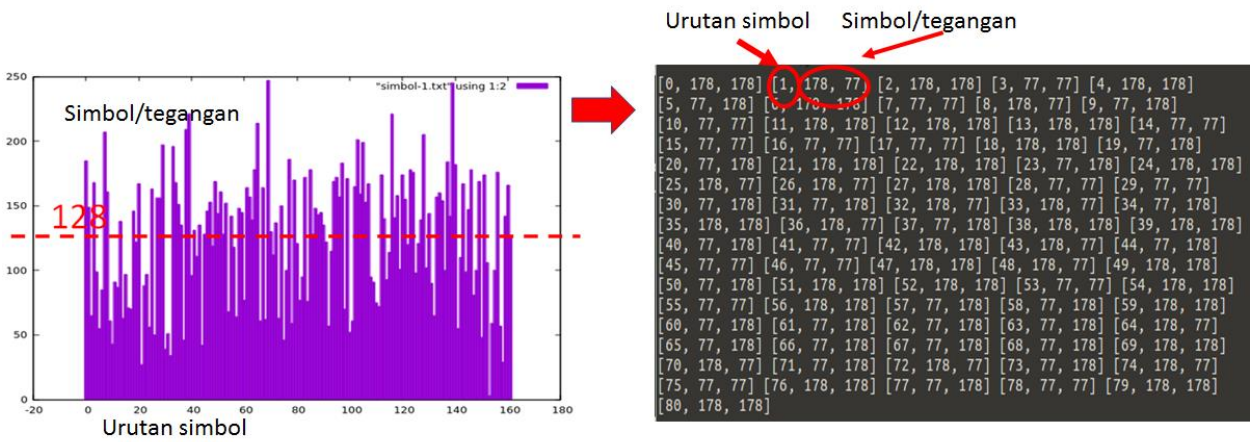
Sebelum malaksanakan Dynamic Progamming, terlebih dahulu prosesor mempersiapkan array sepanjang 81 dengan model parameter **struct** seperti dibawah ini,

struct node {

```
 unsigned long encstate;// state enkoder untuk node selanjutnya (SE)
 long gamma;            // metrik path sekarang (MPS)
 int metrics[4]; // nilai metrik untuk kemungkinan munculnya digit 00,01,10,11
 int tm[2];             // pilihan metrik sekarang setelah konversi
                // encode data menjadi simbol (lsym)
                // komplemen lsym
 int i;                 // branch depan yang sedang diuji
};
```

Terminologi "metrik" dipakai untuk menggambarkan ukuran kedekatan/kemiripan/kemungkinan-kemunculan/loglikelihood suatu input simbol yang diukur terhadap suatu standar/referensi yang sudah diketahui sebelumnya.

Hasil ekstraksi data dari radio penerima, yang sudah dikondisikan menjadi sinyal dengan kuantisasi 8 bit dengan rentang 0 – 255 akan dipasang pada array yang disiapkan dalam **array struct node** yang diberi nama **nodes** sepanjang 80. Contoh hasil penerimaan radio menjadi deretan nilai-nilai yang sudah dikuantisasi 8 bit, yang kemudian kita sebut sebagai deretan simbol dapat dilihat pada Gambar X-7 dibawah ini.

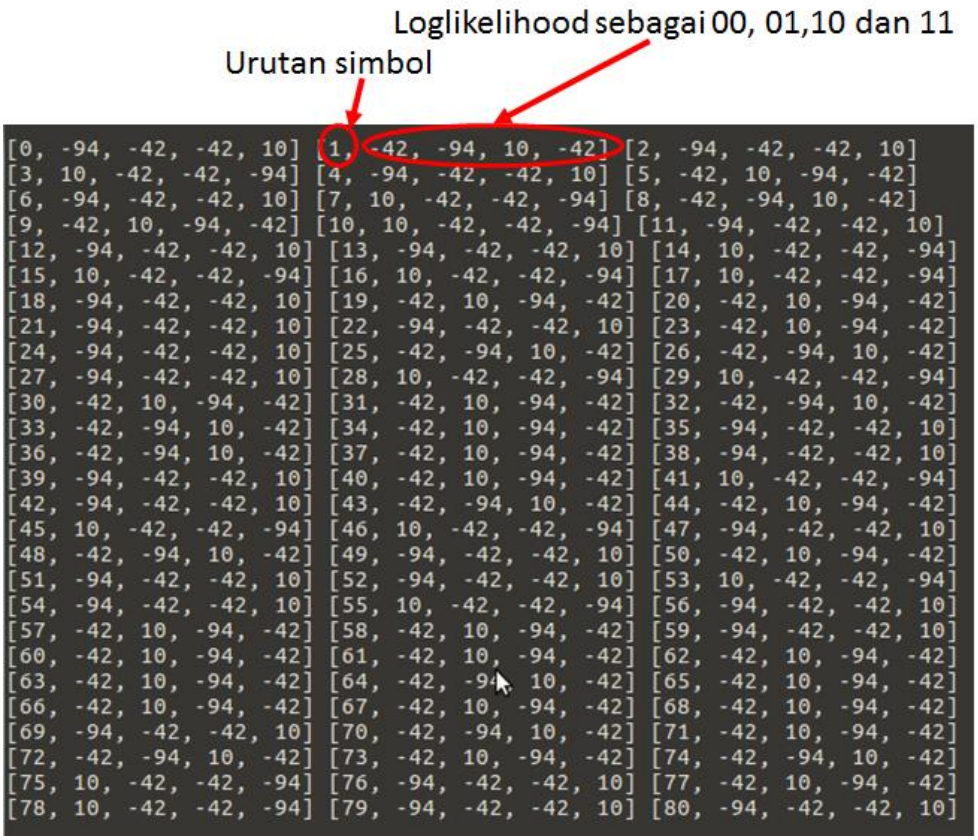


Gambar IX-7: Contoh suatu deretan simbol hasil proses penerimaan radio yang sudah dikondisikan untuk siap didekode.

Deretan simbol-simbol tersebut kemudian diletakkan pada array dengan sintaksis seperti dibawah ini setelah dikonversi menjadi besaran loglikelihood untuk kemungkinan digit-digit 00, 01, 10 atau 11 seperti dibawah ini.

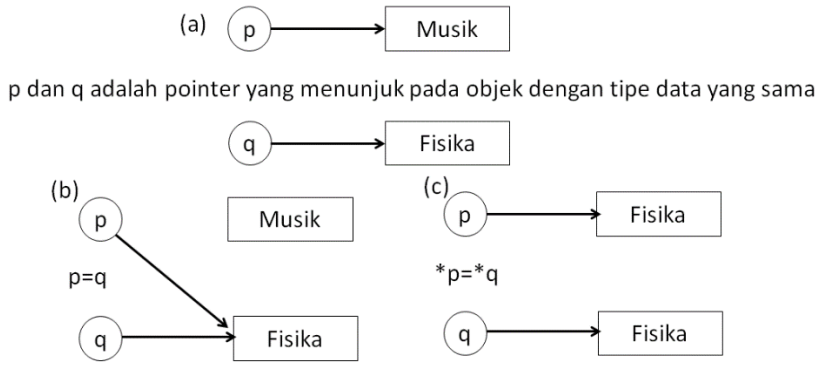
```
struct node {
 unsigned long encstate;      // Encoder state of next node
 long gamma;                  // Cumulative metric to this node
 int metrics[4];              // Metrics indexed by all possible tx syms
```

```
  int tm[2];                    // Sorted metrics for current hypotheses
  int i;                        // Current branch being tested
};
.........
.........
struct node *nodes;     // First node
struct node *np;                // Current node
struct node *lastnode; // Last node
struct node *tail;          // First node of tail
.........
.........
if((nodes = (struct node *)malloc((nbits+1)*sizeof(struct node))) == NULL) {
    printf("malloc failed\n");
    return 0;
  }
.........
.........
lastnode = &nodes[nbits-1];
tail = &nodes[nbits-31];
  *maxnp = 0;
  for(np=nodes;np <= lastnode;np++) {
    np->metrics[0] = mettab[0][symbols[0]] + mettab[0][symbols[1]];
    np->metrics[1] = mettab[0][symbols[0]] + mettab[1][symbols[1]];
    np->metrics[2] = mettab[1][symbols[0]] + mettab[0][symbols[1]];
    np->metrics[3] = mettab[1][symbols[0]] + mettab[1][symbols[1]];
    symbols += 2;   /* pasangan 2 bit konvolusi */
  }
```



Gambar IX-8: Contoh array likelihood yang akan diletakkan pada array struktur int metrics[4]

Dari sintaksis diatas tampak bahwa pada proses inisiasi ini, pointer np dan nodes menunjuk pada nilai nilai array yang sama (sebagai ilustrasi perhatikan Gambar X-9), yang terlebih dahulu sudah dideklarasi dengan pointer nodes untuk deklarasi array dinamik dengan fungsi **malloc()**. Seperti ilustrasi pada Gambar X-9(b), pointer **np** menunjuk pada objek yang juga ditunjuk oleh pointer **nodes** yang sudah dideklarasikan sebagai array dinamik dengan struktur elemen sama dengan pointer **np**.



Gambar IX-9: (a) Pointer p dan q menunjuk pada objek yang berbeda namun mempunyai tipe data yang sama; (b) Pointer p menunjuk pada objek yang sama dengan yang ditunjuk oleh

pointer q, dengan resiko objek yang tadinya ditunjuk oleh pointer p akan hilang; (c) Objek pointer q dipetakan pada objek pointer p

Sebelum melangkah lebih jauh untuk membahas sintaksis dari algoritma Fano, kita bahas terlebih dahulu arti dari parameter metrics[],

- metrics[0] adalah nilai kemungkinan (loglikelihood) simbol diartikan sebagai digit "0" , bersama sama dengan nilai loglikelihood simbol disampingnya diartikan sebagai digit "0" → [00]
- metrics[1] adalah nilai loglikelihood simbol diartikan sebagai digit "0" , bersama sama dengan nilai loglikelihood simbol disampingnya diartikan sebagai digit "1" → [01]
- metrics[2] adalah nilai loglikelihood simbol diartikan sebagai digit "1" , bersama sama dengan nilai loglikelihood simbol disampingnya diartikan sebagai digit "0" → [10]
- metrics[3] adalah nilai loglikelihood simbol diartikan sebagai digit "1" , bersama sama dengan nilai loglikelihood simbol disampingnya diartikan sebagai digit "1" → [11]

Nilai-nilai metrics[] sepanjang 81 nodes inilah yang akan diuji kebenarannya berdasarkan algoritma Fano untuk mengurai kode konvolusi yang sudah dikirim dari sisi pemancar.

X.6 Beberapa Kondisi Yang Akan Diamati

Sebelum mencoba mengamati algoritma Fano, berikut ini beberapa kondisi penting yang harus diamati yaitu:

1. Algoritma akan maju jika metriks path sekarang MPS (np->nodes) ditambah metriks best branch depan MBBD (np->tm[np->i]) lebih besar atau sama dengan threshold (T).
2. Jika MPS + MBBD ≥ T + D (delta) maka untuk T=T+D untuk node berikutnya.
3. Algoritma akan melonggarkan/mengurangi nilai T jika MPS + MBBD < T dan MPS dibelakangnya lebih kecil dari T, maka T=T-D, dan algoritma maju lagi.
4. Jika MPS + MBBD < T dan MPS dibelakangnya lebih lebih besar atau sama dengan nilai T, maka algoritma mundur selangkah, dan m[0] merubah menjadi m[1] (MBBD urutan kedua), jika MPS + MBBD sudah lebih besar T, maka algoritma akan maju lagi.
5. Jika langkah 4 masih gagal, maka algoritma mundur selangkah lagi, sambil diperhatikan apakah m[1] sudah pernah dipakai sebelumnya, kalau sudah maka algoritma mundur lagi mencari node dengan m[1] belum pernah terpakai.
6. Jika langkah mundur sampai pada node 0, maka m[1] kembali ke m[0] dengan nilai T=T-D.
7. Langkah 5 harus selalu disertai dengan langkah 3, jika langkah tiga dimungkinkan maka pada node berikutnya T=T-D dan m[1] dikembalikan ke m[0].
8. Demikian seterusnya sampai alogoritma mencapai node 80, atau upaya langkah mencapai batas maksimum (maxcycle= 10000 × 81). Jika algoritma dapat mencapai node 80, artinya algoritma sukses dalam mendekode sinyal datang. Jika sampai batas upaya maksimum

algoritma melum mencapai node 80, maka algoritma dianggap gagal mendekode sinyal datang.

9. Polinomial yang dipakai sebagai generator kode mempunyai orde 31 (ganjil), sehingga kodeword yang dihasilkan berupa pasangan bit yang komplemen. Keadaan ini dapat dilihat dengan mengamati hasil program encode.c. Sifat komplemen ini kemudian dipakai untuk mempersempit kemungkinan dalam proses menghitung/memilih loglikelihood node didepan.

XI.7 LANGKAH PERCOBAAN

1. Siapkan modul Raspberry Pi, nyalakan dan buka satu atau beberapa terminal window.
2. Masuk ke direktori "PRAKTIKUM-FANO"
3. Kopi beberapa file wav: simbol.WAV, 170224_0732-11.WAV, 170224_0734-20.WAV, 170224_0742-25.WAV, 170224_0730-25.WAV.
4. Kopi file encode.c ke direktori ini.
5. Kompilasi encode.c
%> gcc encode.c -o encode -lm -Wall
6. Jalankan program "encode"
7. Amati hasilnya dengan menekan enter berulang-ulang.
8. Amati hasil konvolusi kodeword yang dibangkitkan saat input encode digit "0".
9. Setelah tekan enter, bandingkan output konvolusi saat input encode digit "1".
10. Catat fenomena ini dan sebutkan pada laporan perihal gejala ini.
11. Pada program "encode" ini sebutkan ada berapa cara yang dipakai untuk membangkitkan codeword.
12. Kompilasi wsprd.c pada direktori ini.
%> make -k
13. Jalankan "wsprd" dengan beberapa file wav yang sudah disiapkan.
%> ./wsprd ./simbol.WAV
Amati saat pergantian nilai threshold (T), serta jumlah upaya.
%> ./wsprd ./170224_0734-20.WAV
Amati saat pergantian nilai threshold (T), serta jumlah upaya.
%> ./wsprd ./170224_0742-25.WAV
Amati pergantian nilai T, saat algoritma mundur, serta jumlah upaya.

14. Buka program fano.c dan beri tanda remarks untuk semua "getchar()", kemudian kompilasi ulang wsprd.c dan fano.c.
%> make -k
15. Jalankan program "wsprd"
%> ./wsprd ./170224_0730-25.WAV \r

Amati sampai program selesai, berapakah angka upaya agar sampai pada akhir proses dekode.

X.8 Laporan Praktikum

1. Jawablah pernyaan 10 dan 11, sertakan kopi sintaksis perihal tersebut.
2. Dapatkan beberapa data dari proses penerimaan sinyal WSPR secara mandiri.
3. Dekode semua data wav yang Saudara dapatkan, tunjukkan fenomena seperti pada percobaan 13, 14 dan 15.
4. Laporan dikumpulkan sebelum PRAKTIKUM 10 dilaksanakan.