UNIVERSITAS INDONESIA

RANCANG BANGUN RANGKAIAN *CONVOLUTIONAL ENCODER* DAN *VITERBI DECODER* MENGGUNAKAN DSK TMS320C6713 BERBASIS SIMULINK

SKRIPSI

MOHAMMAD ABDUL JABBAR
0403030675

FAKULTAS TEKNIK ELEKTRO
PROGRAM STUDI TEKNIK ELEKTRO
DEPOK
JULI 2008

UNIVERSITAS INDONESIA

RANCANG BANGUN RANGKAIAN *CONVOLUTIONAL ENCODER* DAN *VITERBI DECODER* MENGGUNAKAN DSK TMS320C6713 BERBASIS SIMULINK

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik Elektro

MOHAMMAD ABDUL JABBAR
0403030675

FAKULTAS TEKNIK ELEKTRO
PROGRAM STUDI TEKNIK ELEKTRO
DEPOK
JULI 2008

HALAMAN PERNYATAAN ORISINALITAS

Skripsi/Tesis/Disertasi ini adalah hasil karya saya sendiri,

dan semua sumber baik yang dikutip maupun dirujuk

telah saya nyatakan dengan benar.

Nama	**: Mohammad Abdul Jabbar**
NPM	**: 0403030675**
Tanda Tangan	**: .....**
Tanggal	**: 4 Juli 2008**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh :

Nama	: Mohammad Abdul Jabbar
NPM	: 0403030675
Program Studi	: Teknik Elektro
Judul Skripsi	: Rancang Bangun Rangkaian *Convolutional Encoder* dan *Viterbi Decoder* Menggunakan DSK TMS320C6713 Berbasis Simulink

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Elektro, Fakultas Teknik, Universitas Indonesia

DEWAN PENGUJI

Pembimbing : Dr. Ir. Arman D. Diponegoro, MM. (..... tanda tangan .....)

Penguji : Ir. Ny. Rochmah N. Sukardi, M.Sc. (..... tanda tangan .....)

Penguji : Prof. Dr. Ir. Dadang Gunawan, M.Eng. (..... tanda tangan .....)

Ditetapkan di : Depok

Tanggal : 4 Juli 2008

KATA PENGANTAR

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya, saya dapat menyelesaikan skripsi ini. Penulisan skripsi ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar Sarjana Teknik Jurusan Teknik Elektro pada Fakultas Teknik Universitas Indonesia. Saya menyadari bahwa, tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan skripsi ini, sangatlah sulit bagi saya untuk menyelesaikan skripsi ini. Oleh karena itu, saya mengucapkan terima kasih kepada:

(1) Dr. Ir. Arman Djohan Diponegoro, MM. selaku dosen pembimbing yang telah menyediakan waktu, tenaga, dan pikiran untuk mengarahkan saya dalam penyusunan skripsi ini;
(2) Orang tua dan keluarga saya yang telah memberikan bantuan dukungan material dan moral;
(3) Keluarga besar Kamuka Parwata Fakultas Teknik Universitas Indonesia, yang telah membantu dalam memberikan semangat, dan mengajarkan banyak hal kepada penulis selama menjalankan kehidupan kampus di FTUI tercinta;
(4) Devita Anggiarani yang selalu memberikan semangat, dan membantu dalam proses pengetikan skripsi ini;
(5) Dicky Jonathan yang telah mengajarkan teori Matlab, Simulink, dan Code Composer Studio, serta cara menggunakan DSK TMS320C6713;
(4) Sahabat-sahabat yang telah banyak membantu saya dalam menyelesaikan skripsi ini.

Akhir kata, saya berharap Tuhan Yang Maha Esa berkenan membalas segala kebaikan semua pihak yang telah membantu. Semoga skripsi ini membawa manfaat bagi pengembangan ilmu.

Depok, 19 Juni 2008

Penulis

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama : Mohammad Abdul Jabbar

NPM : 0403030675

Program Studi : Teknik Elektro

Departemen : Teknik Elektro

Fakultas : Teknik

Jenis karya : Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty-Free Right*)** atas karya ilmiah saya yang berjudul :

Rancang Bangun Rangkaian *Convolutional Encoder* dan *Viterbi Decoder* Menggunakan DSK TMS320C6713 Berbasis Simulink

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/format-kan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya tanpa meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok

Pada tanggal : 4 Juli 2008

Yang menyatakan

(Mohammad Abdul Jabbar)

ABSTRAK

Nama : Mohammad Abdul Jabbar
Program Studi : Teknik Elektro
Judul : Rancang Bangun Rangkaian *Convolutional Encoder* dan *Viterbi Decoder* Menggunakan DSK TMS320C6713 Berbasis Simulink

Convolutional code merupakan teknik *Error control coding* untuk mendeteksi dan mengkoreksi error pada informasi akibat pengaruh *noise*. Skripsi ini merancang bangun rangkaian *convolutional encoder* dan *Viterbi decoder* menggunakan DSK TMS320C6713 berbasis Simulink, untuk melihat probabilitas error yang dipengaruhi oleh *Binary Symetric Channel* (BSC) sebagai pembangkit error. Analisis meliputi perbandingan bit input dan output dengan variasi nilai parameter *convolutional code*, dan uji coba rangkaian menggunakan DSK TMS320C6713. Hasil penelitian menunjukkan *convolutional encoder* dan *Viterbi decoder* dapat menurunkan probabilitas error tergantung dari parameter yang digunakan. Semakin besar *constrain length* dan semakin kecil *rate* yang digunakan, maka *probability of error* akan semakin kecil.

Kata kunci:
Error control coding, convolutional code, Probability of error, Simulink, DSK TMS320C6713

ABSTRACT

Name : Mohammad Abdul Jabbar
Study Program: Teknik Elektro
Title : Designing Circuitry of Convolutional Encoder and Viterbi Decoder Using DSK TMS320C6713 with Simulink Based

Convolutional code is a technique in *Error control coding* to detect and correct error on information caused by noise. This research designed the circuitry of convolutional encoder and Viterbi decoder using DSK TMS320C6713 with Simulink-based, to see the probability of errors affected by Binary Symetric Channel as error generator. The analysis consists of comparison between input and ouput bits with variation of parameter's value of the convolutional code, and the tryout using DSK TMS320C6713. Results showed that convolutional encoder and Viterbi decoder could reduce the probability of error depend on parameters used. Higher constrain length and smaller rate, resulted a smaller probability of error.

Keywords:
Error control coding, convolutional code, Probability of error, Simulink, DSK TMS320C6713

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

DAFTAR SINGKATAN

ECC	*Error Control Coding*
FEC	*Forward Error Correction*
BSC	*Binary Symmetric Channel*
DSP	*Digital Signal Processing*
DSK	*Digital Signal Processing Kit*
CCS	*Code Composer Studio*
RTDX	*Real Time Data Exchange*
LED	*Light Emitting Diode*
JTAG	*Joint Team Action Group*

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Error control coding adalah teknik untuk mendeteksi dan mengkoreksi *error* pada suatu informasi yang terjadi akibat pengiriman data yang dipengaruhi oleh *noise*. Untuk menganalisis proses pengiriman informasi dan *error* yang terjadi diperlukan sebuah rangkaian yang merupakan skematik dari pengiriman informasi seperti *encoder*, modulasi, saluran transmisi, demodulasi, dan *decoder*.

Salah satu metode dalam *error control coding* adalah *convolutional code* yang dalam skripsi ini menggunakan *convolutional encoder* dan *Viterbi decoder*. Saat ini rangkaian *convolutional encoder* dan *Viterbi decoder* telah diproduksi dalam bentuk hardware/chip yang salah satunya diproduksi oleh Qualcomm. Beberapa penelitian mengenai *error control coding* juga telah dilakukan sebelumnya seperti dalam skripsi oleh Muhammad Suryanegara yang berjudul "*Analisa Konfigurasi Serial Concatenated Code Dengan Menggunakan Teknik Forward Error Correction (FEC) Convolutional Code dan A Posteriory Probability (APP) Decoder* ".

Dengan adanya teknologi DSP prosesor yang terintegrasi dalam sebuah perangkat DSK board TMS320C6713 memungkinkan proses *error control coding* tersebut dapat dibangun dengan rangkaian baseband lainnya dalam satu chip DSK yang dirancang dengan perangkat lunak, dan dengan adanya perkembangan Matlab/Simulink rancangan perangkat lunak dapat dilakukan dengan menggunakan Library dari program Simulink yang tersedia. Sehingga dalam skripsi ini akan dibangun perangkat lunak Simulink dari rangkaian *convolutional encoder* dan *Viterbi decoder* yang nantinya akan dioperasikan ke dalam DSP prosesor dengan menggunakan DSK TMS320C6713.

1.2 Tujuan Penelitian

Tujuan dari skripsi ini adalah merancang bangun rangkaian *convolutional encoder* dan *Viterbi decoder* menggunakan DSK Board TMS320C6713 berbasis simulink, lalu membandingkan informasi yang dikirim dengan informasi yang diterima dengan memberi bit *error* pada saluran komunikasi, yang dalam percobaan ini menggunakan *Binary Symetric Channel*.

1.3 Batasan Masalah

Pembahasan rangkaian *convolutional encoder* dan *Viterbi decoder* ini yang dirancang menggunakan perangkat lunak Simulink dan diuji pada DSK Board TMS320C6713. Percobaan ini tidak menggunakan modulasi dan demodulasi sebagaimana dalam saluran komunikasi yang sesungguhnya. Percobaan ini juga tidak menggunakan pembangkit *error* pada saluran komunikasi yang sudah umum digunakan seperti *noise dari AWGN Channel*. Tetapi dalam percobaan ini menggunakan pembangkit *error* BSC (*Binary Simetric Channel*), merubah bit informasi yang dikirim secara acak untuk melihat kemampuan *decoder* mendeteksi dan mengkoreksi *error*.

1.4 Metodologi Penelitian

Pada penelitian ini menggunakan DSK Board TMS320C6713 sebagai *convolutional encoder* dan *Viterbi decoder* yang sebelumnya dibuat rancang bangun dalam bentuk model menggunakan program Simulink. Kemudian model rancang bangun tersebut dengan MATLAB Simulink akan membangkitkan bahasa mesin melalui program *code composer studio* dan memprogram DSK TMS320C6713 sesuai dengan rancang bangun rangkaian tersebut. Pembahasan dan analisis mengenai perbandingan informasi yang dikirim atau input dengan informasi yang diterima atau output dilakukan dengan menggunakan program simulink dan dengan uji coba pada DSK TMS3206713.

1.5 Sistematika Penulisan

Penulisan makalah skripsi ini dibagi menjadi 5 bab, yaitu

1. Bab 1 Pendahuluan

   Pada bab pertama ini akan dijelaskan mengenai latar belakang yang terkait dengan *error control coding*, tujuan, batasan masalah, metodologi penelitian dan sistematika penelitian.

2. Bab 2 *Convolutional Code* dan DSK TMS3206713 Berbasis Simulink

   Pada bab ini merupakan dasar teori yang menjelaskan tentang *error control coding*, jenis *error, convolutional coding, convolutional encoder, Viterbi decoder, probability of error,* DSK TMS3206713 dan Simulink.

3. Bab 3 Rancang Bangun *Convolutional Encoder* dan *Viterbi Decoder* Menggunakan DSK Board TMS320C6713 Berbasis Simulink.

   Pada bab ini menjelaskan rancang bangun rangkaian c*onvolutional encoder* dan *Viterbi decoder* pada Simulink, implementasi rangkaian *convolutional encoder* dan *Viterbi decoder* pada DSK TMS320C6713.

4. Bab 4 Uji Coba dan Analisis

   Pada bab ini akan menjelaskan tentang hasil uji coba rancang bangun rangkaian *convolutional encoder* dan *Viterbi decoder* menggunakan program Simulink, dan hasil uji coba rangkaian tersebut yang dioperasikan pada DSK TMS320C6713.

5. Bab 5 Kesimpulan

   Isi dari bab ini adalah kesimpulan yang didapat dari hasil penelitian.

BAB 2
CONVOLUTIONAL CODE DAN DSK TMS320C6713 BERBASIS SIMULINK

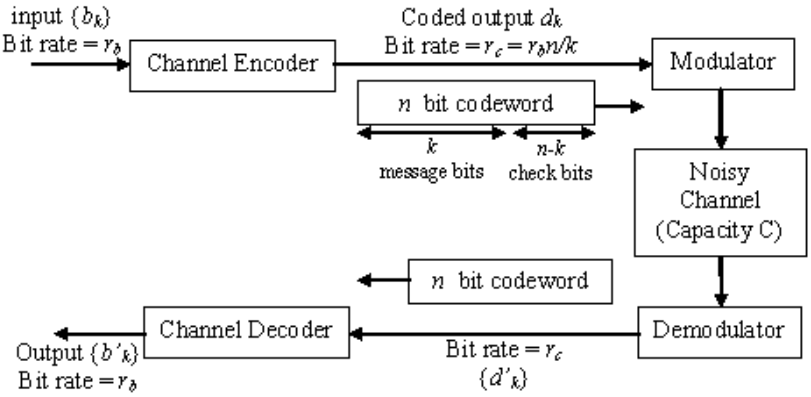
2.1 *Error Control Coding* [1]

Dalam sistem komunikasi digital, *error control coding* merupakan sistem dari pengendali error dalam transmisi data, dimana pengirim(*sender*) mengirimkan data tambahan (*redundant*) ke dalam pesan yang dikirim, yang juga dikenal dengan kode pengkoreksi error. Hal ini dapat membuat penerima(*receiver*) dapat mendeteksi dan mengkoreksi error tanpa perlu meminta pengirim(*sender*) untuk mengirimkan data kembali.

Keuntungan dari *error control coding* ini adalah saluran pengiriman kembali tidak diperlukan atau pengiriman kembali data transmisi dapat dihindari, dalam mempertimbangkan biaya yang dikeluarkan untuk *Bandwidth* yang diperlukan. Jadi *error control coding* digunakan dalam situasi dimana pengiriman data kembali sangat memerlukan biaya.

Probability of Error (Pe) merupakan banyaknya error yang terjadi pada sistem transmisi data yang dirumuskan sebagai perbandingan antara jumlah *bit* yang salah (mengalami *error*) pada sisi penerima dengan jumlah total *bit* yang dikirimkan oleh sisi pengirim. Probabilitas dari sebuah error (P_e) pada skema pengiriman sinyal dapat juga dikatakan merupakan fungsi dari *Signal to Noise Ratio* (SNR) pada input penerima(*receiver*) dengan tingkat kecepatan data (*rate*).

Fungsional blok yang dapat melaksanakan *error control coding* adalah *channel encoder* dan *channel decoder* seperti yang ditunjukkan pada gambar 2.1 di bawah ini.



Gambar 2.1 Skema dari *error control coding* [1]

Channel encoder menambahkan digit pada digit pesan yang dikirim. Digit tambahan ini tidak membawa informasi baru bagi pesan yang dikirim dan juga membuat mungkin *channel decoder* untuk mendeteksi dan mengkoreksi error pada digit informasi yang dikirim. Dengan menambahkan *encoder* dan *decoder* pada pengiriman informasi ini maka pendeteksian dan atau pengkoreksian error tersebut diharapkan dapat menurunkan probabilitas error secara keseluruhan.

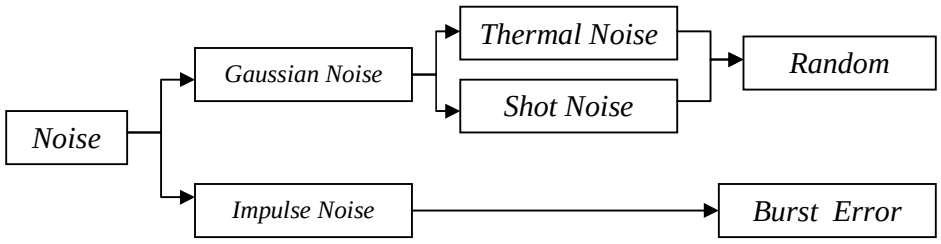
Dengan melihat Gambar 2.1 di atas, sistem komunikasi digital untuk keluaran biner dari sumber *encoder* $\{b_k\}$ melewati saluran yang ber-*noise* pada tingkat kecepatan r_b bit/detik. Dalam permasalahan *channel noise*, aliran bit (*bit stream*) $\{b'_k\}$ dipenuhi oleh penerima terkadang berbeda dari urutan yang ditransmisikan $\{b_k\}$, karena itu diinginkan agar probabilitas error $P(b'_k \neq P_k)$ bernilai kurang dari nilai yang sudah ditetapkan. Data transmisi yang aktual yang melewati saluran dapat terlaksana oleh modem (modulator dan demodulator) yang dapat beroperasi pada tingkat data r_c memiliki jangkauan $r_1 \leq r_c \leq r_2$. Probabilitas error q_c untuk modem, didefinisikan akan bergantung pada tingkat kecepatan data r_c dalam saluran komunikasi tersebut. Metode *error control coding* akan mengurangi probabilitas (P_e) dari *error* secara keseluruhan dengan beberapa aspek-aspek berikut:

a. *Error control coding* dapat mendeteksi dan mengkoreksi *error* dengan menambahkan bit tambahan yang disebut juga *check bits* pada pesan yang akan dikirim.

b. *Error control coding* tidak dapat mendeteksi dan mengkoreksi semua *error*.
c. *Check bits* mengurangi *rate* dari data yang dikirimkan melalui saluran komunikasi. Efisiensi dari *rate* didefinisikan sebagai r_b/r_c.

2.2 Jenis *Noise* [1]

Dalam sistem komunikasi digital, *error* pada transmisi disebabkan oleh *noise* dalam saluran komunikasi. Secara umum, dalam saluran komunikasi dapat dibedakan dua jenis *noise*, seperti diilustrasikan pada Gambar 2.2 di bawah ini



Gambar 2.2 Dua macam jenis noise yang mucul dalam saluran komunikasi

2.2.1 *Gaussian Noise*

Tipe yang pertama adalah *Gaussian noise*. Sumber-sumber *Gaussian noise* antara lain adalah *thermal noise* dan *shot noise* pada peralatan pengirim dan penerima, *thermal noise* pada saluran, dan radiasi yang diterima oleh antena penerima. Kepadatan spektrum daya pada *Gaussian noise* pada input *receiver* seringkali berwarna putih. *Error* transmisi yang dibawa oleh *Gaussian noise* putih adalah *error* yang kemunculannya selama interval pensinyalan tertentu tidak mempengaruhi performa sistem selama interval pensinyalan *subsequent*. *Error* transmisi yang diakibatkan oleh *Gaussian noise* putih disebut juga sebagai *random error*.

2.2.2 *Impulse Noise*

Tipe kedua *noise* yang seringkali ditemukan dalam saluran komunikasi adalah *impulse noise*, dengan karakternya berupa interval panjang yang sunyi diikuti oleh *noise* dengan amplitudo tinggi. Tipe *noise* ini seringkali disebabkan oleh kejadian alami atau buatan manusia seperti misalnya petir atau *switching*

sementara. Ketika *noise* muncul maka akan mempengaruhi lebih dari satu simbol atau *bit*, dan biasanya terdapat dependensi *error* dalam simbol-simbol yang dikirimkan secara berurutan. Hal ini menyebabkan *error* muncul secara tiba-tiba.

Skema pengendalian *error* untuk mengatasi *random errors* disebut sebagai kode-kode *random error-correcting*, dan skema pengkodean yang dirancang untuk memperbaiki *burst errors* disebut kode-kode *burst error correcting*.

2.3 *Convolutional Code* [1]

Dalam *convolutional encoder* n digit kode yang dihasilkan oleh encoder dalam unit waktu tidak hanya tergantung dari blok k digit dari pesan yang dikirim pada waktu tersebut, tetapi juga tergantung dari blok pesan yang dikirim sebelumnya. *Convolutional code* dapat dirancang untuk mendeteksi dan mengkoreksi error. Proses *encoding* dari *convolutional code* dapat tercapai menggunakan *shift register*, dan beberapa prosedur praktis dalam proses pen-dekodean juga telah dikembangkan.

2.3.1 *Convolutional Encoder*

Convolutional encoder, seperti ditunjukkan pada Gambar 2.3, dihasilkan dengan melewatkan urutan informasi yang ditransmisikan melalui shift register terdiri dari tiga parameter yang bernilai integer yaitu[2] :

k = jumlah bit input yang akan dilewatkan ke dalam shift register

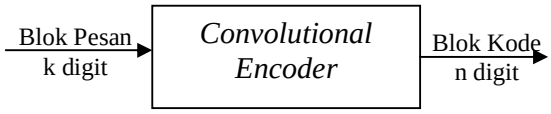
n = jumlah bit output decoder

K = constrain length

dengan *code rate* atau laju pengkodean data adalah

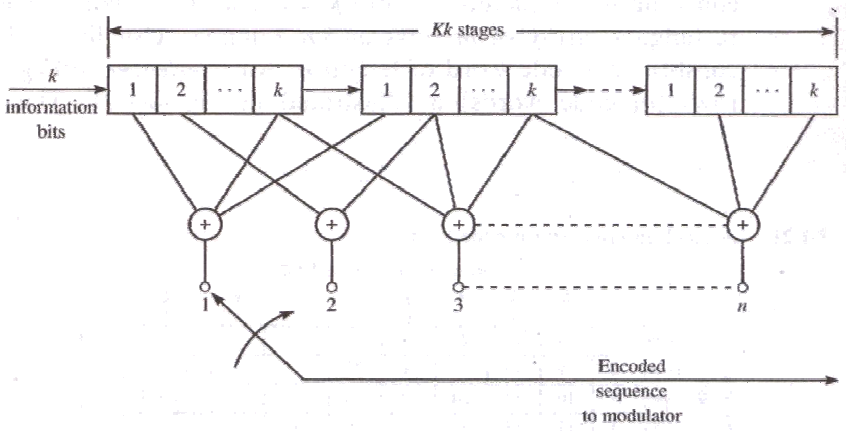
$$r = k/n \qquad \textbf{(2.1)}$$

Dalam unit waktu pada *convolutional encoder*, blok pesan terdiri atas *k* digit dimasukkan ke dalam *encoder* lalu *encoder* menghasilkan kode blok yang terdiri atas *n* digit kode (k<n). Blok *n* digit kode tidak hanya tergantung dari *k* digit pesan pada waktu yang sama, tapi juga tergantung dari dari blok pesan yang sebelumnya[1].



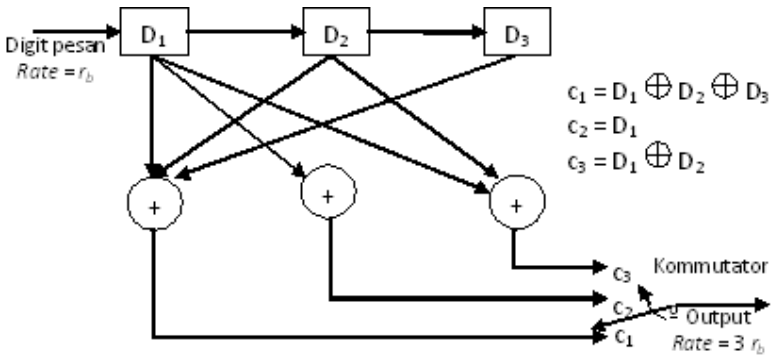
Gambar 2.3 Skema c*onvolutional encoder* [1]

Kode yang dihasilkan pada *encoder* di atas disebut (n,k) *convolutional code* dengan *constrain length K* dijit dan kecepatan menghasilkan data (*rate*) *k/n*[1]. Secara umum shift register terdiri dari K(*k-bit*) stage dan *n* bit output. Input data yang masuk ke dalam encoder merupakan data biner, lalu *k* bit digeser ke dalam shift register pada satu waktu seperti yang ditunjukkan pada Gambar 2.4 di bawah ini[2].



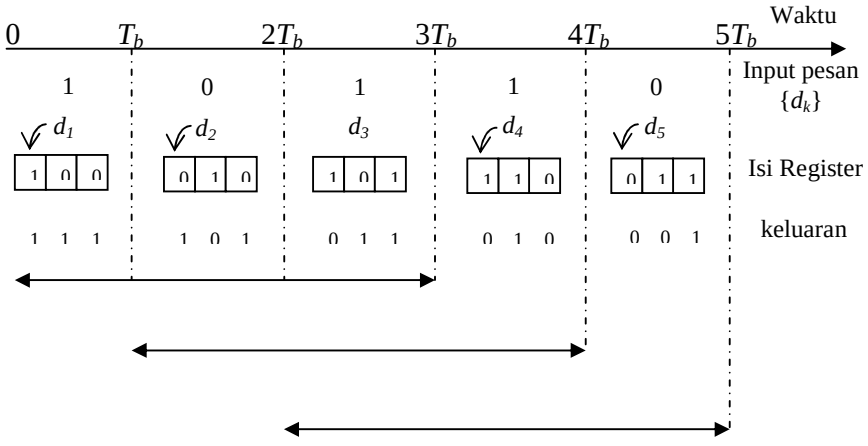
Gambar 2.4 *Shift register* c*onvolutional encoder* [2]

Dapat dicontohkan untuk *input bit* 10110 setelah di-*encode* menjadi 111101011010001. Hasil ini diperlihatkan pada sebuah *encoder convolutional code* dengan k=1, n=3, K=3 yang ditunjukkan pada Gambar 2.5 di bawah ini.[1]



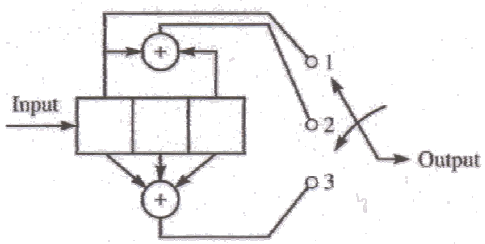
Gambar 2.5 *shift register* c*onvolutional encoder*, *K*=3, *n*=3, *k*=1 [1]

Operasi dari *encoder* ditunjukkan pada Gambar 2.5 dilakukan seperti pada gambar. Kita asumsikan bahwa *shift register* memiliki nilai awal yang kosong. Bit pertama dari input data dimasukkan ke D_1. Selama interval bit pesan komutator mencuplik *modulo-2 adder* dengan output c_1, c_2, dan c_3. Maka bit pesan tunggal menghasilkan tiga bit output. Pada pesan berikutnya urutan input kini memasuki D_1, sementara isi dari D_1 digeser ke D_2 dan komutator kembali mencuplik tiga *adder* pada output. Proses kembali berulang sampai digit pesan terakhir digeser ke dalam D_3. Sebagai contoh dari operasi *encoding* ditunjukkan dalam Gambar 2.6 di bawah ini.

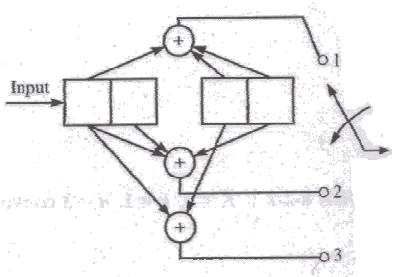


Gambar 2.6 Operasi *encoding* pada *convolutional encoder* yang ditunjukkan pada gambar 2.5 [1]

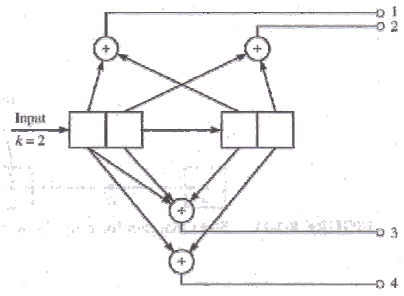
Berikut ini merupakan contoh dari shift register dengan ukuran yang berbeda yang ditunjukkan oleh Gambar 2.7, Gambar 2.8, dan Gambar 2.9 di bawah ini:



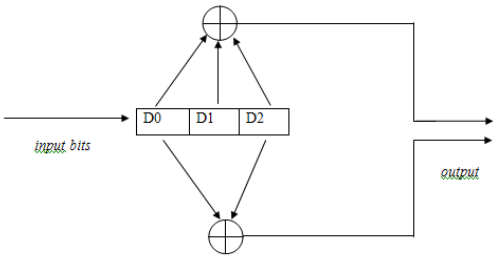
Gambar 2.7 *Shift register convolutional encoder* dengan K=3, k=1, n=3 [2]



Gambar 2.8 *Shift register convolutional encoder* dengan dengan K=2, k=2, n=3 [2]



Gambar 2.9 *Shift register convolutional encoder* dengan K=2, k=2, n=4 [2]



Gambar 2.10 *Encoder Convolutional code* , K=3 dan r=1/2 [3]

2.3.1.1 Generator Polynomial [2]

Salah satu metode untuk menggambarkan *convolutional code* adalah dengan matriks generator. Setiap vektor memiliki ukuran dimensi *Kk* dan mengandung garis hubungan pada encoder menuju *modulo 2 adder*. Masukan nilai "1" pada posisi i (baris) dari vector untuk garis hubungan pada shift register yang menuju *modulo 2 adder* dan masukan "0" pada posisi vector jika tidak ada hubungan antara *shift register* dengan *modulo 2 adder*.

Sebagai contoh pada Gambar 2.7 di atas dengan $K = 3$, $k = 1$, $n = 3$. Pada kondisi awal semua shift register berisi nol. Generator menghasilkan urutan output setiap tiga bit pada output 1, 2 dan 3, maka generatornya adalah

G1 = [100]

G2 = [101]

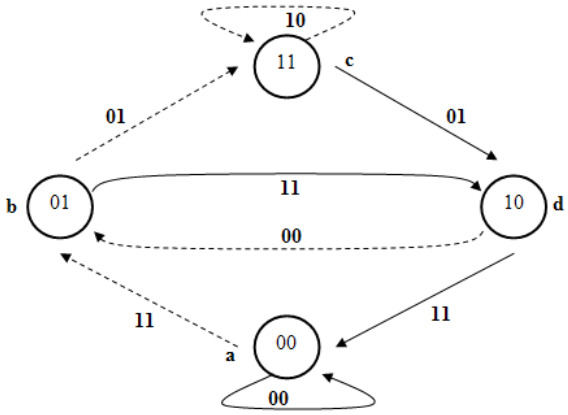
G3 = [111]

Generator pada kode ini biasanya ditulis dalam bentuk oktal dengan biner 100 adalah nilai oktal dari 4, biner 101 adalah nilai oktal dari 5, dan biner 111 adalah nilai oktal dari 7. Sehingga generator polynomial yang sudah diubah dalam bentuk oktal menjadi (4, 5, 7).

2.3.1.2 State Diagram [3]

Proses *encoder* dapat direpresentasikan oleh *state diagram*. Gambar 2.11 menunjukkan diagram untuk spesifikasi *encoder* pada Gambar 2.10. Angka dalam lingkaran menunjukkan 2 *bit* awal pada isi *register*, *bit* kedua menyatakan *bit* yang baru masuk. *Input bit* '0' ditandai oleh garis penuh (*solid line*) sedangkan *input bit* '1' ditandai dengan garis putus-putus (*dash line*). Dua *bit* pada kedua garis tersebut menyatakan *bit output* pada interval waktu tertentu.

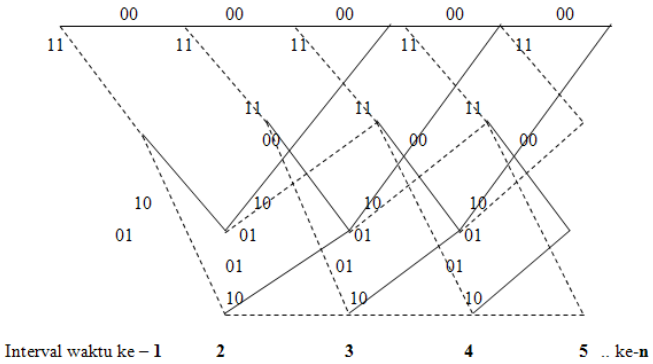
Sebagai contoh, untuk kondisi awal *register* 00 maka jika diberi *input* 0 kondisi *register* akan menjadi 00 (tetap) dan *output encoder* adalah 00, sebaliknya jika diberi *input* 1 kondisi *register* akan berubah menjadi 01 dan *output encoder* adalah 11.



Gambar 2.11 *State Diagram* untuk K=4 dan r=1/2 [3]

2.3.1.3 Trellis Diagram [3]

Trellis diagram juga merupakan salah satu cara untuk merepresentasikan proses *encoder*. Diagram ini biasanya digunakan untuk memudahkan dalam proses *decoding*. Gambar 2.12 menunjukkan skema *trellis diagram* untuk rangkaian pada Gambar 2.10. *Input encoder* untuk *bit* '1' ditandai dengan garis putus-putus (*dash line*) sedangkan *input bit* '0' ditandai dengan garis penuh (*solid line*). *Output encoder* pada interval waktu tertentu dinyatakan oleh nilai 2 *bit* yang berada pada garis tersebut. Dari Gambar 2.2 tampak bahwa untuk *input* 0 atau 1 pada interval waktu ke-**n** dimana n > K , *encoder* akan menghasilkan *output* antara 00 atau 11, 11 atau 00, 10 atau 01, dan 01 atau 10. *Output* yang dihasilkan tergantung pada *bit input* sebelumnya, yaitu saat interval waktu ke **n-1**.



Gambar 2.12 *Trellis diagram* untuk *encoder* K=2 dan r =1/3 [3]

2.3.2 *Convolution Decoder* Dengan Algoritma *Viterbi*

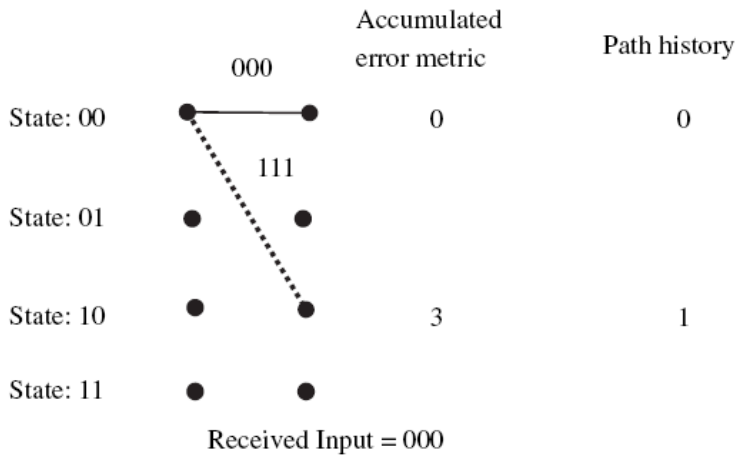
Pada tahun 1967, *Viterbi* memperkenalkan sebuah algoritma *decoding* untuk *convolutional encoder* yang saat ini dikenal dengan algoritma *Viterbi*[4]. Algoritma *Viterbi* menunjukkan maximum likelihood *decoding*. *Viterbi* algoritma menggunakan diagram trellis dari *convolutional code* dimana sifatnya digunakan untuk mengimplementasikan algoritma ini. Permasalahan utama dari maximum likelihood adalah jumlah kalkulasi yang harus dilaksanakan dalam urutan kode yang memungkinkan. Algortima *Viterbi* mengurangi kompleksitas dari perhitungan ini dengan menghindari menjadi bagian dari urutan yang mungkin diterima oleh *encoder*[5]. Proses *decoding* menggunakan Algoritma *Viterbi* adalah sebagai berikut[6].

1. Menambahkan: pada setiap siklus, *branch metric* menambahkan satu per satu dari node (state) dengan yang sebelumnya.
2. Membandingkan : jalur metric yang menuju ke keadaan encoder dibandingkan
3. Memilih : jalur *likelihood* tertinggi (*survivor*) yang menuju ke keadaan encoder dipilih, sedangkan yang terendah dibuang.

Untuk lebih memahami pengertian di atas akan dijelaskan sebagai berikut.

a) *Hamming distance d* adalah jumlah bit akatau karakter yang berbeda antara dua vektor yang dibandingkan, misal *Hamming Distance* antara 101 dengan 100 adalah 2. Terdapat dua bit yang berbeda pada masing-masing posisi vektor di atas.
b) *Branch Metric* adalah *Hamming Distance* antara simbol yang diterima atau input *Viterbi decoder* dengan semua simbol yang mungkin diterima.
c) *Branch metric* dapat juga dikatakan sebagai *error metric*.
d) *Error metric* dari tiap jalur pada *trellis* akan diakumulasikan tiap waktu dengan *error metric* sebelumnya.
e) Jika terdapat dua jalur yang tiba pada satu *node* atau simpul yang sama pada *trellis*, maka akan dipilih jalur dengan nilai *error metric* terendah.
f) *Path History* merupakan jalur trellis dari *branch metric* yang sedang mengalami proses akumulasi *error metric*.

Sebuah encoder dengan *constrain length K* = 2, *k = 1*, dan *n* = 3. Pada waktu *t*=1 urutan simbol yang diterima pada input *decoder* adalah 000, dan simbol yang yang mungkin diterima pada saat *t*=1 adalah 000 dan 111. Maka nilai *branch metric* antara 000 dengan 000 atau *state* 00 dengan *state* 00 adalah 0 dan nilai *branch metric* antara 000 dengan 111 atau *state* 00 dengan *state* 10 adalah 3, karena waktu *t*=1 merupakan waktu pertama maka akulumasi *error metric* merupakan nilai *error metric* pada saat itu juga yang merupakan nilai dari *branch metric*. Gambar 2.13 di bawah ini merupakan ilustrasi dari proses tersebut.

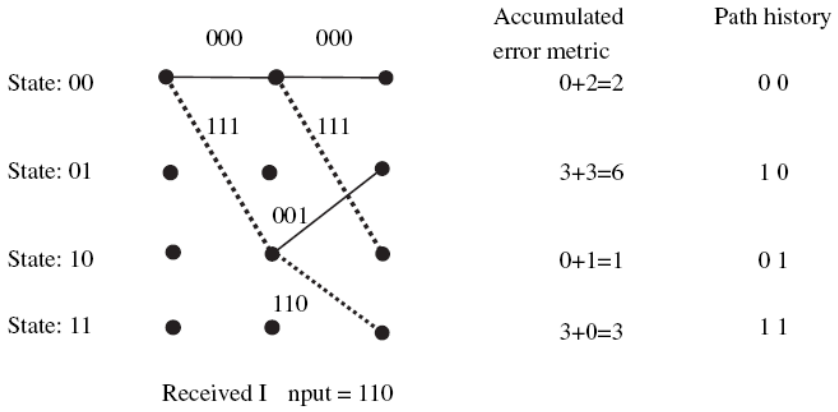


Gambar 2.13 Proses *decoding* dengan algortima *Viterbi* saat *t*=1 [6]

Path history pada gambar di atas menunjukkan jalur trellis yang dilewati dari *node* atau simpul yang terakumulasi. *Path history* pertama menunjukkan nilai 0 karena jalur melewati garis menyambung (*solid line*) yang merupakan representasi dari input 0, oleh karena itu *path history* bernilai 0, dan begitu pula dengan *Path history* yang kedua yang bernilai 1 karena melewati garis putus-putus yang merupakan representasi dari input 1.

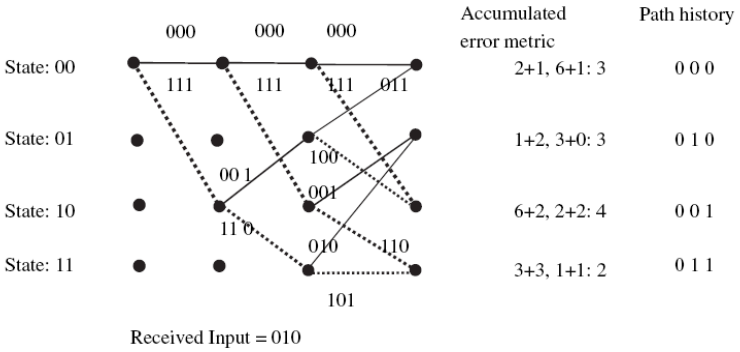
Pada saat *t*=2 simbol input yang diterima oleh *encoder* adalah 110, dan simbol yang mungkin diterima pada waktu kedua ini yaitu 000, 111, 001, dan 110. Kemudian akan dibandingkan *Hamming distance* antara masing-masing simbol yang mungkin diterima tersebut dengan simbol yang diterima oleh input *decoder*, sehingga menghasilkan nilai *error metric* yang diakumulasikan dengan nilai *error metric* pada jalur sebelumnya. *Path history* menunjukkan jalur yang sudah

dilewati oleh *branch metric* dari *node* pertama yaitu pada waktu *t*=1. Gambar 2.14 di bawah ini menunjukkan ilustrasi dari proses *decoding* saat *t*=2.



Gambar 2.14 Proses *decoding* dengan algortima *Viterbi* saat *t*=2 [6]

Saat *t*=3 simbol input yang diterima adalah 010 dan pada waktu ketiga ini simbol yang mungkin diterima sebanyak delapan simbol yaitu 000, 111, 011, 100, 001, 110, 010, dan 110. Proses perhitungan sama dengan proses sebelumnya yaitu menghitung *hamming distance* lalu mengakumulasikan nilai *error metric* dengan yang sebelumnya. *Path history* bertambah menjadi tiga bit yang menunjukkan proses telah melewati tiga jalur *branch metric*. Gambar 2.15 di bawah ini menunjukkan ilustrasi dari proses *decoding* saat *t*=3.



Gambar 2.15 Proses *decoding* dengan algortima *Viterbi* saat *t*=3 [6]

Pada *t*=3 ini terdapat dua jalur yang tiba pada *node* yang sama, dan pada saat inilah yang akan dilakukan perbandingan. Jalur yang akan memiliki nilai akumulasi *error metric* terendah merupakan jalur yang akan dipilih, kemudian

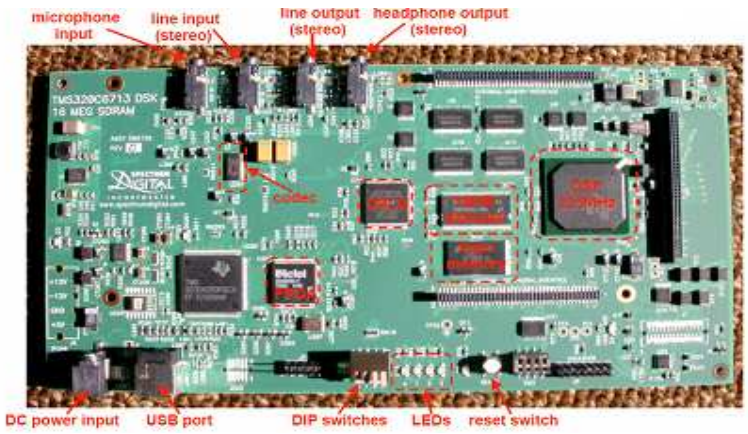
disebut sebagai jalur *survivor*, sedangkan jalur yang lain akan dibuang. Kemudian pada *t*=3 ini pula terlihat dari Gambar 2.15 di atas bahwa *Path History* sudah terisi penuh yaitu berjumlah tiga bit. Pada saat ini akan dibandingkan dari setiap *state* yang memiliki nilai akumulasi *error metric* terendah. Jalur yang memiliki nilai akumulasi *error metric* terendah disebut dengan jalur *final survivor*, yang menentukan output dari *Viterbi decoder*. Output dari *Viterbi decoder* adalah nilai *Left Significant Bit* atau nilai bit yang paling kiri dari *Path History* pada jalur *final survivor*. Proses ini akan berulang sampai simbol bit terakhir yang diterima oleh *decoder*, sehingga didapatkan keseluruhan output dari *Viterbi decoder*.

2.4 DSK TMS320C6713

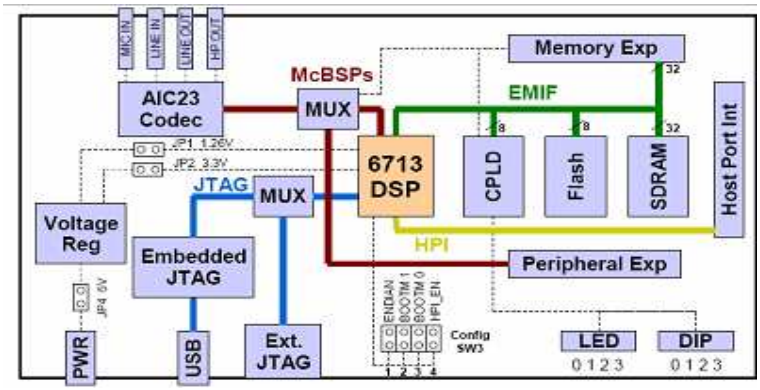
DSK (Digital Signal Processor Kit) merupakan perangkat yang dibuat oleh *Texas Instrument* dan salah satu jenis produknya adalah TMS320C6x merupakan mikroprosesor yang memiliki arsitektur khusus dan sekumpulan instruksi yang dikhususkan untuk pemrosesan sinyal secara *real time*. Prosesor TMS320C6713 memiliki kecepatan yang cukup tinggi dan penggunaan yang cukup baik pada serangkaian proses komputasi numerik yang kompleks. Digital Signal Processor ini dapat digunakan untuk aplikasi yang besar, dari komunikasi dan kendali hingga pemrosesan suara dan gambar. Tujuan umum dari DSP ini didominasi oleh aplikasi dalam komunikasi seperti selular. Aplikasi yang menggunakan DSP ini pada umumnya dipakai dalam produk-produk konsumen seperti telepon selular, fax/modem, disk drive, MP3 Player, radio, printer, alat bantu dengar, High Definition television (HDTV), kamera digital.

Pemrograman DSK dapat dilakukan dengan fasilitas perangkat lunak (*software*) yang dikenal dengan nama Code Composer Studio (CCS). Basis bahasa pemrograman yang digunakan perangkat lunak ini yaitu bahasa C dan C++. Proses instalasi program oleh komputer ke memori tertentu pada DSK (*loading*) dilakukan dengan menggunakan *port* USB yang merupakan bagian dari modul JTAG. Selain menggunakan CCS secara langsung, proses pembuatan kode program dapat dilakukan secara tidak langsung dengan menggunakan Matlab Simulink versi 7.4.0.

Berikut ini adalah gambar dari papan elektronik C6713 DSK yang ditunjukkan oleh Gambar 2.16 Sedangkan blok diagram modul-modulnya ditunjukkan oleh Gambar 2.17.



Gambar 2.16. Papan C6713 DSK dengan bagian-bagiannya.



Gambar 2.17. Blok diagram papan C6713 DSK.

TMS320 C6713 DSK memiliki beberapa modul input dan output yang diantaranya adalah:

a. LINE IN dan LINE OUT

LINE IN merupakan modul input yang mengubah sinyal analog menjadi sinyal diskrit dengan frekuensi cuplik tertentu. Sedangkan LINE OUT merupakan modul output yang mengubah sinyal diskrit menjadi sinyal kontinu yang terinterpolasi. Jenis proses interpolasi sinyal yang digunakan adalah *Zero Order Hold*. Frekuensi cuplik yang digunakan pada modul LINE IN maupun

LINE OUT cukup bervariasi, kisaran nilai dimulai dari 8 kHz, 32 kHz, 44,1 kHz hingga yang tertinggi yakni 96 kHz. Penggunaan kedua modul ini (termasuk pula modul MIC IN sebagai modul input dan HP OUT sebagai modul output) melibatkan *codec* AIC23.

b. LED dan DIP Switch

Modul LED merupakan modul output yang sinyal outputnya direpresentasikan dalam bentuk penyalaan LED. Sedangkan DIP Switch merupakan modul input yang sinyal inputnya direpresentasikan dalam bentuk perubahan tegangan hasil dari penekanan (*switching*) tombol DIP Switch.

c. JTAG (*Joint Team Action Group*)

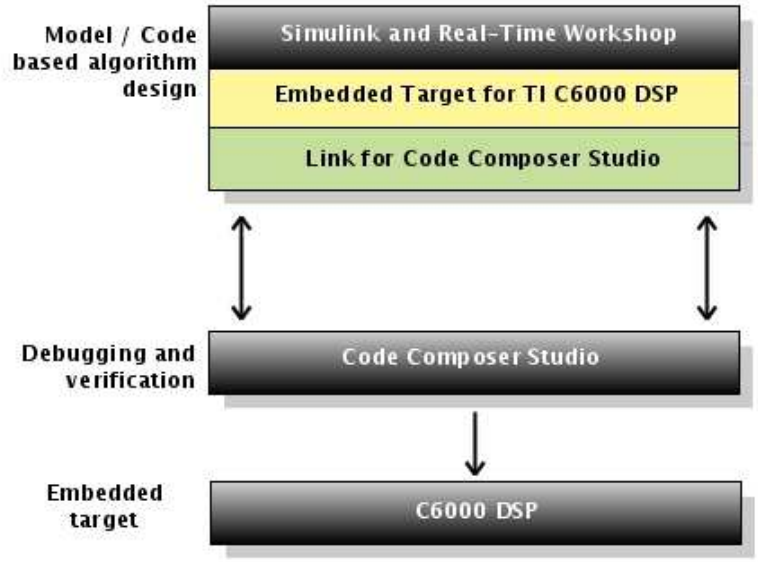
Modul ini memungkinkan terjadinya komunikasi antara DSK dengan komputer. Modul ini merupakan modul input sekaligus modul output. Melalui *port* USB (*Universal Serial Bus*), pertukaran data dan kontrol serta pengawasan proses pengeksekusian program dapat dilakukan antara DSK dengan komputer. Proses pertukaran data secara *real time* antara DSK dengan komputer dengan menggunakan JTAG dikenal dengan nama RTDX (*Real Time Data Exchange*). Proses RTDX dilakukan dengan cukup intensif karena proses ini memerlukan *timing* yang teratur dan tepat antara komputer dengan DSK serta sinkronisasi diantara keduanya. Jika dibandingkan dengan kecepatan cuplik pada modul LINE IN dan modul LINE OUT, maka kecepatan rata-rata dari proses pertukaran sebuah data terjadi lebih lambat.

2.5 Simulink dan *Code Composer Studio*

Simulink merupakan sub program dari perangkat lunak Matlab 7.4.0 yang berfungsi untuk merancang suatu model yang akan digunakan untuk proses simulasi. Modul atau sub program tersebut dapat diaktifkan dengan mengetikkan kata ”simulink” pada *command window* Matlab 7.4.0. Dalam *library* simulink terdapat blok-blok yang dapat mewakili setiap fungsi dari rancangan yang akan dibuat dalam bentuk model. Model yang disimulasikan dapat berupa model fisik, model matematis, dan model logika (*logic*). Simulink secara otomatis dapat membangkitkan kode-kode bahasa pemrograman seperti bahasa C atau C++

sehingga model rancangan dapat diprogram ke dalam DSK TMS320C6713 sehingga dapat dilakukan uji coba menggunakan DSK tersebut.

Pembangkitan kode-kode program dan pemrograman prosesor DSP tersebut dilakukan secara otomatis oleh Simulink bekerja sama dengan perangkat lunak CCS (Code Composer Studio ™). Alur pemrograman prosesor TMS320C6713 ditunjukkan oleh Gambar 2.18 berikut ini. Perangkat lunak CCS mengubah (meng-*compile*) kode-kode bahasa C atau C++ yang dibangkitkan oleh Simulink menjadi bahasa mesin dari prosesor TMS320C6713.

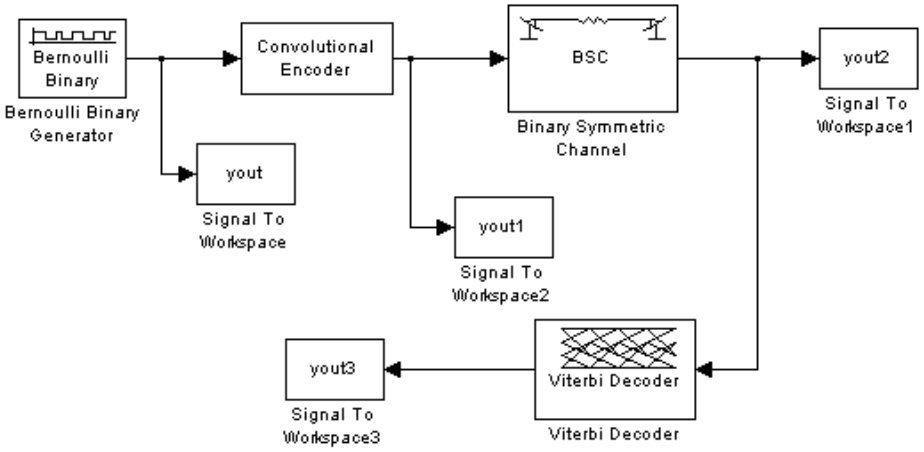


Gambar 2.18 Alur instalasi algoritma model ke dalam memori C6713 DSK sebagai *Embedded Target.*

BAB 3
RANCANG BANGUN RANGKAIAN *CONVOLUTIONAL ENCODER* DAN *VITERBI DECODER* MENGGUNAKAN DSK TMS320C6713 BERBASIS SIMULINK

3.1 Rancang Bangun Rangkaian *Convolutional Encoder* dan *Viterbi Decoder* pada Program Simulink

Untuk mengimpelementasikan rangkaian *convolutional encoder* dan *viterbi decoder* ke dalam DSP prosesor yang terdapat dalam DSK320C6713, maka sebelumnya dibuat rancang bangun dan mensimulasikannya menggunakan perangkat lunak Simulink. Pada Gambar 3.1 di bawah ini menunjukkan skema rangkaian pengiriman informasi yang menggunakan *convolutional encoder* dan *viterbi decoder* dengan sumber data berupa biner.



Gambar 3.1 Rancang bangun *convolutional encoder* dan *viterbi decoder* menggunakan program simulink.

Gambar 3.1 di atas merupakan rancangan yang akan digunakan lalu disimulasikan dalam program Simulink untuk melihat proses yang terjadi dan error yang diperoleh dari metode *error control coding* ini. Rancang bangun ini menggunakan masukan berupa data biner yang acak seperti dapat dilihat pada

gambar di atas menggunakan *bernoulli binary random generator*. Lalu data melewati *convolutional encoder* kemudian melewati saluran komunikasi yang dipengaruhi oleh error yang menggunakan *Binary Symetric Channel*. Lalu setelah sinyal data mengalami error pada saluran transmisi kemudian melewati *viterbi decoder* yang berfungsi untuk mendeteksi dan mengkoreksi error. Sinyal output yang telah melwati *viterbi decoder* kemudian ditampilkan melalui blok *Signal to Workspace*.

Berikut ini akan dijelaskan blok-blok yang digunakan pada rancang bangun diatas beserta parameter yang digunakan pada blok tersebut.

a) Blok *Bernoulli Binary Generator* merupakan blok yang menghasilkan bilangan biner secara acak menggunakan distribusi *bernoulli*. Parameter blok yang digunakan yaitu:

1. *probability of zero* diset menjadi 0,5 artinya kemungkinan blok ini mengeluarkan nilai antara 1 dan 0 adalah 50 %.
2. *initial seed* yaitu pola acak dari distribusi *bernoulli* diset menjadi 61, nilai ini tidak berpengaruh terhadap proses *encoding* dan *decoding*, hanya mengluarkan data biner “1” atau “0” dengan pola acak yang ditentukan.
3. *sample time* bernilai 1, sampel data akan muncul tiap 1 detik
4. *output data type* diset *double*, merupakan panjang atau lebar data.

b) Blok *Convolutional encoder* merupakan blok yang menghasilkan *convolutional code* dari data biner yaitu melakuka*n encoding* dari urutan input biner vektor menjadi urutan output biner vektor. Parameter yang digunakan dalam blok ini yaitu:

1. *Trellis Structure* merupakan dari parameter dari *encoder* atau *decoder* yaitu menunjukkan nilai *constrain length K*, input *k*, dan input *n*. *Trellis Structure* memiliki parameter *constrain length K*, dan Generator matriks dalam bentuk oktal dari *convolutional encoder*. Cara mencari nilai generator matriks sudah dijelaskan pada Bab 2. Nilai *trellis structure* yang digunakan dalam percobaan ini diubah-ubah sesuai dengan parameter *K, k,* dan *n* yang diinginkan dan akan dijelaskan pada Bab 4.

2. *operation mode* yang diset *continuous*. Proses yang terjadi dari *encoder* ini berjalan secara berkelanjutan.

c) Blok *Binary Symetric Channel* merupakan blok yang berfungsi untuk menambahkan error biner pada sinyal input dengan probabilitas tertentu dan bit error ditambahkan secara acak. Parameter yang digunakan dalam blok ini yaitu.

1. *error probability*, probabilitas untuk menambahkan error pada data biner yang masuk, yaitu data biner yang sudah di-*encode*. Nilai *error probability* yang digunakan pada percobaan ini diubah-ubah untuk melihat perbedaan data yang dapat dikoreksi oleh *decoder* dengan perbedaan banyak error yang terjadi pada data yang sudah di-*encode*.
2. *initial seed* yang merupakan pola acak dari bit error yang diberikan. Nilai *initial seed* dalam percobaan ini diubah-ubah agar error yang diberikan tersebar secara acak dan berbeda-beda polanya dalam tiap percobaan, dengan nilai parameter *constrain length K*, *k*, dan *n* yang berbeda-beda.

d) Blok *Viterbi decoder* merupakan blok yang men-*decode* secara convolutional data yang sudah di-*encode* menggunakan *Viterbi* algoritma. Blok *Viterbi decoder* men*decode* simbol input lalu menghasilkan simbol output biner. Blok ini dapat memproses beberapa simbol pada satu waktu untuk performansi yang lebih cepat. Parameter yang digunakan dalam blok ini yaitu:

1. *Trellis Structure* dengan nilai yang berbeda-beda dalam tiap percobaan yang akan disebutkan pada Bab 4.
2. *decition type* diset Hard Decision, berarti data yang dapat keluar dari blok ini hanya data yang bernilai “0” atau “1”.
3. *traceback depth* diset bernilai 1. Delay yang terjadi pada keluaran *decoder* sebesar 1 sample tertinggal terhadap input pada *encoder*.
4. *operation mode* diset *continuous.*

e) Blok *signal to workspace* merupakan blok yang berfungsi untuk menuliskan input ke dalam *workspace* sehingga hasil dari simulasi dapat ditampilkan.

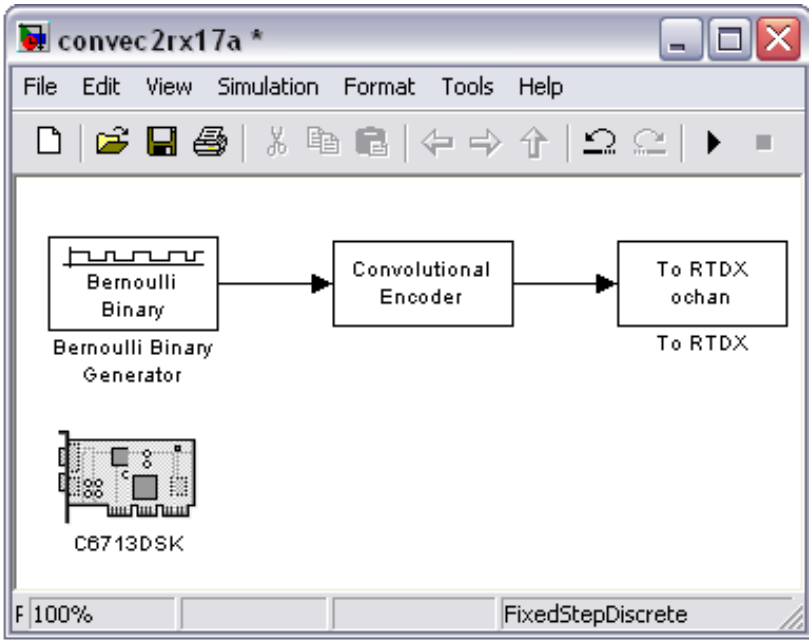
3.2 Implementasi Rangkaian *Convolutional Encoder* ke Dalam DSK TMS320C6713

Rangkaian *convolutional encoder* dan *viterbi decoder* yang dibahas pada subbab sebelumnya merupakan rangkaian keseluruhan yang akan dipisahkan dalam dua rangkaian yaitu *convolutional encoder* dan *viterbi decoder*. Untuk melihat proses uji coba rangkaian *convolutional encoder* dalam DSK TMS320C6713 dan hasilnya ditampilkan ke dalam komputer melalui *RTDX Channel*. Selanjutnya hasil dari proses *encoding* tersebut disimpan ke dalam *workspace* untuk diteruskan ke dalam program Simulink yaitu melewati bit *error* oleh *Binary Symetric Channel* kemudian *Viterbi decoder*. Rangkaian *convolutional encoder* akan diimplementasikan ke dalam DSK TMS320C6713 dengan cara membuat model rangkaian tersebut dalam progam simulink lalu memprogramnya ke dalam DSK TMS320C6713.

Gambar 3.2 di bawah ini merupakan gambar konfigurasi untuk menghubungkan komputer dengan DSK TMS320C6713.

Gambar 3.2 Konfigurasi hubungan PC Komputer dengan DSK TMS320C6713

Untuk melihat hasil yang terjadi pada rangkaian *convolutional encoder* yang diimplemetasikan ke dalam DSK TMS320C6713 ini menggunakan input biner yang dihasilkan dari *bernoulli binary random generator* dan output *RTDX Channel* sehingga dapat ditampilkan ke dalam komputer. Gambar 3.3 di bawah ini menunjukkan rangkaian *convolutional encoder* dalam model Simulink yang diprogram ke dalam DSK TMS320C6713.



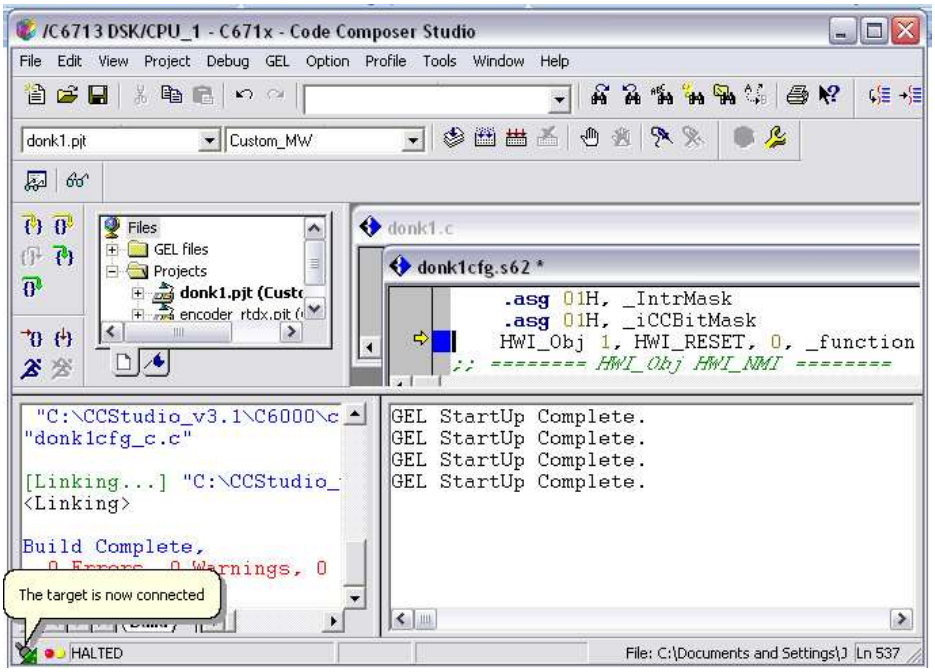
Gambar 3.3 Model rangkaian *convolutional encoder* yang diimplemetasikan ke dalam DSK TMS320C6713.

Gambar 3.3 di atas merupakan model rangkaian *convolutional encoder* yang akan diimplementasikan ke dalam mikroprosesor DSK TMS320C6713. Blok yang digunakan adalah *Bernoulli binary random generator, convolutional encoder, to RTDX,* dan blok *Target Preferences C6713DSK.*

a) Blok *Target Preferences C6713DSK* merupakan blok yang menghubungkan program simulink dengan program *code composer studio* untuk melakukan proses pemrograman dari model Simulink ke dalam DSK TMS320C6713.
b) Blok *to RTDX Channel* merupakan blok yang berfungsi sebagai saluran pengirim data dari target DSK ke dalam komputer sehingga output yang dihasilkan oleh *convolutional encoder* ini dapat ditampilkan ke dalam komputer.

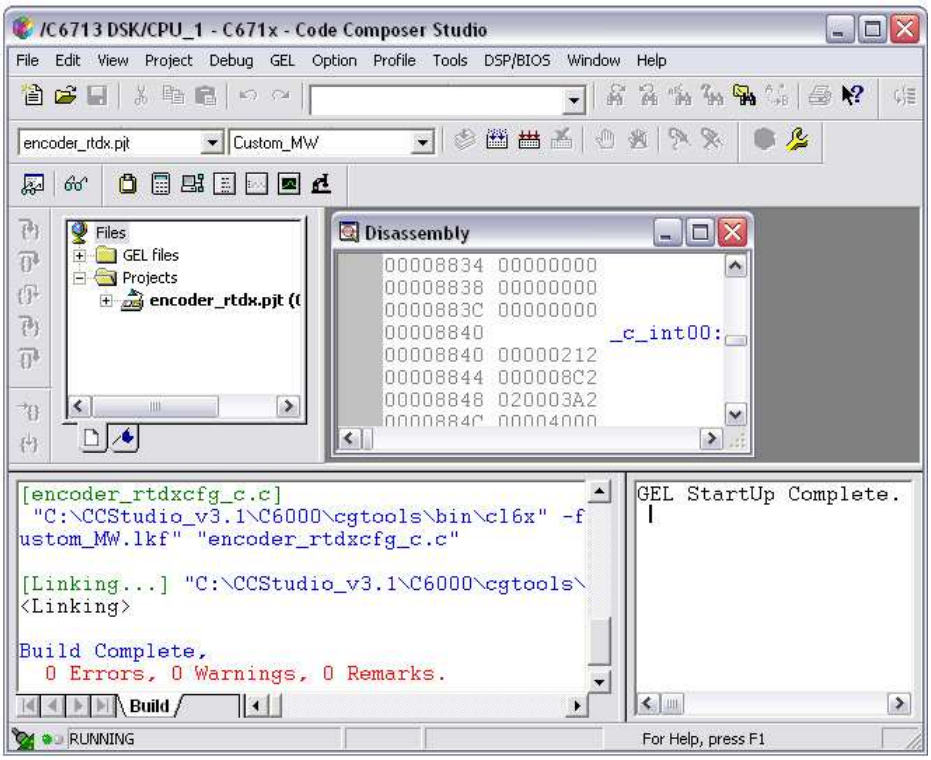
Model rangkaian *convolutional encoder* diimplementasikan atau diprogram ke dalam DSK TMS320C6713 menggunakan tools *incremental build*  yang terdapat dalam simulink, dengan sebelumnya menggunakan *software Code Composer Studio* sebagai pengenal (*driver*) untuk menghubungkan antara

DSK TMS320C6713 dengan PC komputer. Setelah Code Composer Studio menyatakan bahwa DSK TMS320C6713 dan PC komputer sudah terhubung seperti yang ditunjukkan pada Gambar 3.4 di bawah ini, maka model rangkaian *convolutional encoder* siap diprogram ke dalam DSK tersebut.



Gambar 3.4 Tampilan Code Composer Studio yang menyatakan DSK TMS320C6713 sudah terhubung dengan PC komputer.

Setelah *code composer studio* menyatakan bahwa DSK dan komputer sudah terhubung, maka langkah selanjutnya adalah melakukan proses pembangkitan program pada simulink ke dalam DSK TMS320C6713 dengan menekan tombol *incremental build*. Setelah proses pemrograman ke dalam DSK selesai, MATLAB akan memberitahukan bahwa proses telah selesai dan juga pada *code composer studio* seperti yang ditunjukkan Gambar 3.5 di bawah ini merupakan tampilan *code composer studio* yang menyatakan proses pembentukan program telah selesai.

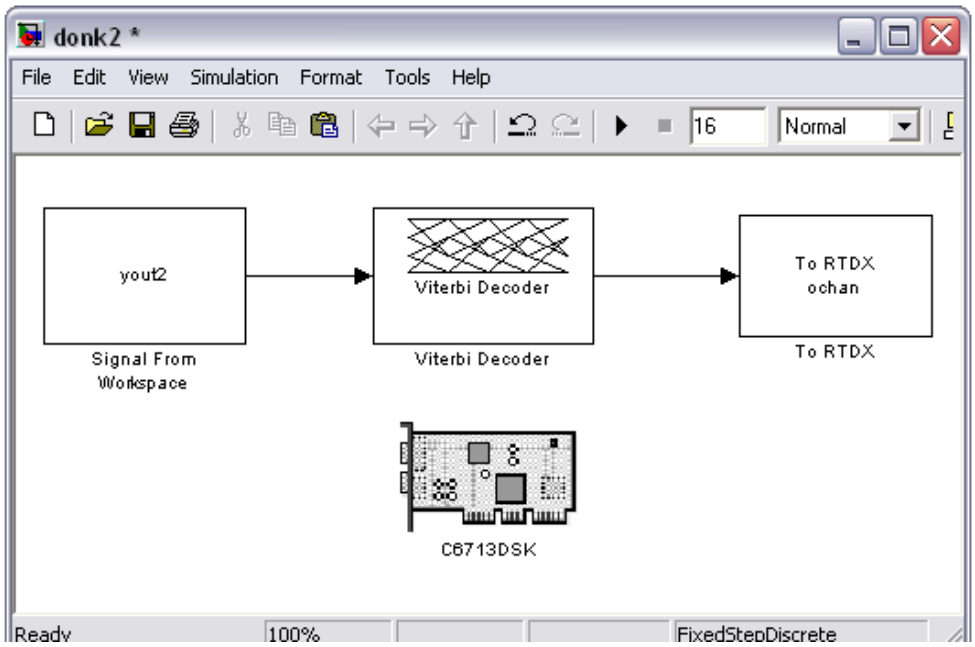


Gambar 3.5 Tampilan Code Composer Studio yang menyatakan proses pembentukan program ke dalam DSK TMS320C6713 telah selesai.

3.3 Implementasi Rangkaian *Viterbi Decoder* ke Dalam DSK TMS320C6713

Pada rangkaian *Viterbi decoder* yang akan dioperasikan ke dalam DSK TMS320C6713 sehingga dapat dilakukan proses uji coba dan menampilkan hasilnya ke dalam komputer melalui *RTDX Channel*. Hasil tersebut merupakan hasil dari keseluruhan rangkaian *convolutional encoder* dan *vierbi decoder* ini, karena pada input *Viterbi decoder* dalam DSK TMS320C6713 merupakan data biner yang berasal dari proses *encoding* dan penambahan bit *error* oleh *Binary Symetric Channel*. Hasil tersebut disimpan di dalam workspace dan menjadi input dari *Viterbi decoder* dalam DSK TMS320C6713.

Proses yang dilakukan untuk mengimplementasikan rangkaian *viterbi decoder* ke dalam DSK TMS320C6713 sama dengan proses mengimplementasikan rangkaian *convolutional encoder* ke dalam DSK TMS320C6713. Pertama yang dilakukan adalah membuat model rangkaian *viterbi decoder* di dalam simulink seperti yang ditunjukkan oleh Gambar 3.6 berikut ini.



Gambar 3.6 Model rangkaian *viterbi decoder* yang diintegrasikan ke dalam mikroprosesor DSK TMS320C6713.

Model yang dibuat dalam program Simulink tersebut lalu dibangkitkan menjadi program ke dalam DSK TMS320C6713 dengan cara yang sama seperti sub bab sebelumnya. Setelah proses pembentukan program selesai maka DSK TMSC6713 telah berisi dengan program rangkaian *Viterbi decoder* dan siap untuk dilakukan uji coba.

BAB 4
UJI COBA DAN ANALISIS

Pembahasan yang dilakukan pada bab ini yaitu melakukan simulasi rangkaian *convolutional encoder* dan *viterbi decoder* pada program Simulink dengan berbagai nilai parameter dari *encoder* dan *decoder* tersebut dengan menambahkan error pada saluran komunikasi. Error yang ditambahkan ke dalam bit yang telah di-*encode* merupakan pembangkit error yang berasal dari Blok *Binary Symetric Channel* dengan parameter blok yang diubah-ubah untuk perbedaaan bit yang mengalami error. Parameter *encoder* dan *decoder* yang digunakan dalam percobaan ini adalah nilai *constrain length K* = 3, 4, dan 7, nilai input *k* = 1, 2, dan nilai output *n* = 2, 3, dan 4. Parameter blok yang digunakan oleh *convolutional encoder* dan *viterbi decoder* merupakan nilai *constrain length K* dan bentuk oktal dari generator matriks G1, G2, G3, dan G4.

Tabel 4.1 di bawah ini merupakan hasil perhitungan generator matriks dari parameter input *k* dan ouput *n* yang digunakan oleh *encoder* dan *decoder*.

Tabel 4.1 Nilai generator polynomial untuk berbagai *K*, *k*, dan *n*

K	*k*	*n*	Generator (biner)				Generator (oktal)
			G1	G2	G3	G4	
3	1	2	111	101	-	-	(7, 5)
3	1	3	100	101	111	-	(4, 5, 7)
4	2	3	1011	1101	1010	-	(13, 15, 12)
4	2	4	1010	0101	1110	1001	(12, 5, 16, 11)
7	1	2	1111001	1011011	-	-	(171, 133)
7	1	3	1111001	1011011	1010010	-	(171, 133, 122)

Pembahasan juga dilakukan dengan melakukan uji coba rangkaian *convolutional encoder* dan *viterbi decoder* pada DSK TMS320C6713. Uji coba yang dilakukan yaitu dengan konfigurasi DSK DSK TMS320C6713 sebagai

convolutional encoder dan komputer yang dijalankan dalam program Simulink sebagai *viterbi decoder*. Perbandingan input informasi dan output dilakukan hanya pada satu nilai parameter dari *encoder* dan *decoder* saja karena dalam uji coba DSK hanya akan dilihat bahwa DSK TMS320C6713 dapat bekerja dengan baik sebagai *convolutional encoder* dan *viterbi decoder*.

4.1 Percobaan dengan *Constrain Length K*=3

4.1.1 Percobaan dengan Input *k*=1 dan Output *n*=2

Pada percobaan ini menggunakan nilai *constrain length K* = 3, input *k*=1 dan output *n=2* maka generator matriksnya dalam bentuk oktal adalah (7, 5) seperti yang ditunjukkan pada Tabel 4.1 di atas. Parameter blok dari *encoder* dan *decoder* yaitu *Trellis stucture* diset menjadi *poly2trellis(3, [7 5])*. Dalam setiap satu bit input *encoder* akan menghasilkan dua bit *redundant* sehingga terdapat tiga bit pada output dari *encoder*. Percobaan ini menggunakan data input biner yang berasal dari *Bernoulli binary random generator* sebanyak 16 bit atau 2 Byte sehingga dengan spesifikasi *encoder* maka input 16 bit akan di*encode* menjadi dua kalinya yaitu 32 bit yang berisi 16 bit informasi asli dan 16 bit *redundant*. Hasil percobaan menunjukkan sebagai berikut. Bit input yang dihasilkan dari *Bernoulli binary random generator* ditunjukkan pada Tabel 4.2 di bawah ini.

Tabel 4.2 Bit input rangkaian *convolutional encoder* dan *viterbi decoder* pada simulink

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1

4.1.1.1 Percobaan dengan 5 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,25 dan *initial seed* = 33. Tabel 4.3 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.4 merupakan perbandingan bit input dengan bit output.

Tabel 4.3 Perbandingan bit yang di-*encode* dengan menambahkan 5 bit error pada *K*=3, *k*=1, *n*=2

No	bit yang telah di-*encode*		bit mengalami error	
1	1	1	1	1
2	1	0	1	0
3	0	0	0	0
4	0	1	0	**0**
5	0	1	**1**	1
6	0	0	0	1
7	0	1	0	1
8	1	0	1	0
9	0	1	0	1
10	1	1	1	1
11	0	0	0	**1**
12	1	1	1	1
13	0	1	**1**	1
14	0	1	**1**	1
15	0	0	0	0
16	0	1	0	1

Tabel 4.4 Perbandingan bit input dengan bit output dengan *K*=3, *k*=1, *n*=2 dan 5 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	**0**	**1**	1	1	1	0	0	0	1	**0**	0	1	**0**

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 4/16

= 0,25

4.1.1.2 Percobaan dengan 10 bit error pada urutan yang di-encode

Tabel 4.5 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error sebanyak 10 bit dan Tabel 4.6 merupakan perbandingan bit input dengan bit output.

Tabel 4.5 Perbandingan bit yang di-*encode* dengan menambahkan 10 bit error pada *K*=3, *k*=1, *n*=2

No	bit yang telah di-*encode*		bit mengalami error	
1	1	1	1	1
2	1	0	**0**	**1**
3	0	0	0	0
4	0	1	0	1
5	0	1	0	1
6	0	0	0	0
7	0	1	0	**0**
8	1	0	1	0
9	0	1	**1**	1
10	1	1	**0**	1
11	0	0	0	0
12	1	1	1	**0**
13	0	1	0	1
14	0	1	**1**	**0**
15	0	0	0	**1**
16	0	1	0	**0**

Tabel 4.6 Perbandingan bit input dengan bit output dengan *K*=3, *k*=1, *n*=2 dan 10 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	**1**	1	1	0	**0**	1	**0**	**1**	**1**	**1**	1	1	**1**	**0**	1

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 8/16

= 0,5

4.1.1.3 Percobaan dengan 18 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* 0,6 dan *initial seed* 521. Tabel 4.7 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.8 merupakan perbandingan bit input dengan bit output.

Tabel 4.7 Perbandingan bit yang di-*encode* dengan menambahkan 18 bit error pada *K*=3, *k*=1, *n*=2

No	bit yang telah di-*encode*		bit mengalami error	
1	1	1	**0**	1
2	1	0	**0**	0
3	0	0	0	0
4	0	1	**1**	**0**
5	0	1	**1**	1
6	0	0	0	0
7	0	1	**1**	**0**
8	1	0	**0**	0
9	0	1	**1**	1
10	1	1	1	**0**
11	0	0	**1**	**1**
12	1	1	1	**0**
13	0	1	**1**	**0**
14	0	1	**1**	**0**
15	0	0	0	0
16	0	1	0	**0**

Tabel 4.8 Perbandingan bit input dengan bit output dengan *K*=3, *k*=1, *n*=3 dan 18 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	**0**	0	**0**	**0**	0	1	**0**	**0**	**1**	0	0	1	1	0	**0**	**0**

Probability of error = banyaknya bit yang error = 8/16

banyaknya bit input

= 0,5

4.1.2 Percobaan dengan Input *k*=1 dan Output *n*=3

Pada percobaan ini menggunakan nilai *constrain length K* = 3, input *k*=1 dan output *n=3* maka generator matriksnya dalam bentuk oktal adalah (4, 5, 7). Parameter blok dari *encoder* dan *decoder* yaitu *Trellis stucture* diset menjadi *poly2trellis(3, [4 5 7])*. Dalam setiap satu bit input *encoder* akan menghasilkan dua bit *redundant* sehingga terdapat tiga bit pada output dari *encoder*. Input biner sebanyak 16 bit atau 2 Byte akan di-*encode* menjadi tiga kalinya yaitu 48 bit yang berisi 16 bit informasi asli dan 32 bit *redundant*.

4.1.2.1 Percobaan dengan 6 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* 0,25 dan *initial seed* 71. Tabel 4.9 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.10 merupakan perbandingan bit input dengan bit output.

Tabel 4.9 Perbandingan bit yang di-*encode* dengan menambahkan 6 bit error pada *K*=3, *k*=1, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	1	1
2	0	0	1	0	0	1
3	1	0	0	1	0	0
4	1	1	0	1	1	0
5	0	1	0	0	1	0
6	1	0	0	1	0	0
7	1	1	0	1	1	0
8	1	0	1	1	0	1
9	0	1	0	**1**	1	0
10	0	1	1	0	1	**0**
11	0	0	0	0	0	0
12	1	1	1	1	**0**	1
13	1	1	0	1	1	0
14	0	1	0	0	1	**1**
15	1	0	0	1	**1**	**1**
16	1	1	0	1	1	0

Tabel 4.10 Perbandingan bit input dengan bit output dengan *K*=3, *k*=1, *n*=3 dan 6 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	1	0	1	1	1	**1**	0	0	1	1	0	1	1

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 1/16

= 0,0625

4.1.2.2 Percobaan dengan 12 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* 0,235 dan *initial seed* 212. Tabel 4.11 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.12 merupakan perbandingan bit input dengan bit output.

Tabel 4.11 Perbandingan bit yang di-*encode* dengan menambahkan 12 bit error pada *K*=3, *k*=1, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	**0**	1
2	0	0	1	**1**	0	**0**
3	1	0	0	**0**	0	**1**
4	1	1	0	1	1	0
5	0	1	0	0	1	**1**
6	1	0	0	1	0	**1**
7	1	1	0	1	1	0
8	1	0	1	1	0	1
9	0	1	0	0	1	0
10	0	1	1	0	1	1
11	0	0	0	0	**1**	0
12	1	1	1	**0**	1	1
13	1	1	0	1	1	0
14	0	1	0	**1**	1	0
15	1	0	0	1	0	0
16	1	1	0	1	**0**	**1**

Tabel 4.12 Perbandingan bit input dengan bit output dengan *K*=3, *k*=1, *n*=3 dan 12 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	**1**	1	**0**	0	1	1	1	0	0	0	1	1	0	1	**0**

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 3/16

= 0,1875

4.1.2.3 Percobaan dengan 26 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* 0,54 dan *initial seed* 75. Tabel 4.13 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.14 merupakan perbandingan bit input dengan bit output.

Tabel 4.13 Perbandingan bit yang di-*encode* dengan menambahkan 26 bit error pada *K*=3, *k*=1, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	1	1
2	0	0	1	**1**	**1**	1
3	1	0	0	1	**1**	0
4	1	1	0	1	**0**	**1**
5	0	1	0	**1**	**0**	**1**
6	1	0	0	1	**1**	**1**
7	1	1	0	1	**0**	**1**
8	1	0	1	**0**	0	**0**
9	0	1	0	**1**	**0**	**1**
10	0	1	1	**0**	1	**0**
11	0	0	0	**1**	**1**	**1**
12	1	1	1	**0**	**0**	1
13	1	1	0	1	1	0
14	0	1	0	0	1	0
15	1	0	0	1	0	0
16	1	1	0	1	**0**	**1**

Tabel 4.14 Perbandingan bit input dengan bit output dengan *K*=3, *k*=1, *n*=3 dan 26 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	**1**	1	1	**1**	1	1	1	**1**	0	**1**	1	1	0	1	**0**

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 5/16

= 0,3125

4.2 Percobaan dengan *Constrain Length K*=4

4.2.1 Percobaan dengan Input *k*=2 dan Output *n*=3

Pada percobaan ini menggunakan nilai *constrain length K* = 4, input *k*=2 dan output *n*=3 maka generator matriksnya dalam bentuk oktal adalah (13, 15, 12). Parameter blok dari *encoder* dan *decoder* yaitu *Trellis stucture* diset menjadi *poly2trellis(4, [13 15 12])*.

4.2.1.1 Percobaan dengan 4 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* 0,25 dan *initial seed* 71. Tabel 4.15 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.16 merupakan perbandingan bit input dengan bit output.

Tabel 4.15 Perbandingan bit yang di-*encode* dengan 4 bit yang mengalami error pada *K*=4, *k*=2, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	1	1
2	0	1	0	0	1	0
3	0	1	0	**1**	**0**	0
4	0	1	1	0	1	1
5	1	1	1	1	1	1
6	1	0	0	1	0	0
7	0	1	1	0	1	1
8	0	0	0	0	0	0
9	0	0	1	0	0	1
10	0	1	1	0	1	1
11	1	1	0	1	1	0
12	1	1	1	1	1	1
13	1	0	1	1	0	1
14	1	1	1	**0**	1	1
15	1	0	0	1	0	0
16	0	1	1	0	1	**0**

Tabel 4.16 Perbandingan bit input dengan bit output dengan *K*=4, *k*=2, *n*=3 dan 4 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 0/16

= 0

4.2.1.2 Percobaan dengan 10 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* 0,1875 dan *initial seed* 53. Tabel 4.17 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.18 merupakan perbandingan bit input dengan bit output.

Tabel 4.17 Perbandingan bit yang di-*encode* dengan menambahkan 10 bit error pada *K*=4, *k*=2, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	1	**0**
2	0	1	0	0	1	0
3	0	1	0	**1**	1	0
4	0	1	1	**1**	1	**0**
5	1	1	1	1	1	1
6	1	0	0	1	0	0
7	0	1	1	0	1	1
8	0	0	0	0	0	0
9	0	0	1	**1**	0	1
10	0	1	1	**1**	**0**	1
11	1	1	0	1	1	0
12	1	1	1	1	1	1
13	1	0	1	1	0	**0**
14	1	1	1	1	1	1
15	1	0	0	**0**	0	0
16	0	1	1	0	**0**	1

Tabel 4.18 Perbandingan bit input dengan bit output dengan *K*=4, *k*=2, *n*=3 dan 10 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	**0**	**1**	1	1	1	0	**1**	0	1	1	**1**	1	1

Probability of error = banyaknya bit yang error = 4/16
banyaknya bit input
= 0,25

4.2.1.3 Percobaan dengan 21 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* 0,48 dan *initial seed* 236. Tabel 4.19 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.20 merupakan perbandingan bit input dengan bit output.

Tabel 4.19 Perbandingan bit yang di-*encode* dengan 21 bit yang mengalami error pada *K*=4, *k*=2, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	**0**	1
2	0	1	0	0	1	0
3	0	1	0	0	**0**	**1**
4	0	1	1	0	**0**	1
5	1	1	1	1	**0**	**0**
6	1	0	0	**0**	0	0
7	0	1	1	**1**	1	**0**
8	0	0	0	0	0	**1**
9	0	0	1	0	**1**	1
10	0	1	1	0	1	**0**
11	1	1	0	1	1	**1**
12	1	1	1	**0**	**0**	1
13	1	0	1	**0**	**1**	**0**
14	1	1	1	1	**0**	1
15	1	0	0	**0**	0	**1**
16	0	1	1	0	1	1

Tabel 4.20 Perbandingan bit input dengan bit output dengan *K*=4, *k*=2, *n*=3 dan 21 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	**0**	1	**1**	1	1	**0**	0	0	**1**	1	1	**1**	**0**	**0**

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 7/16

= 0,4375

4.2.2 Percobaan Dengan Input *k*=2 dan Output *n*=4

Pada percobaan ini menggunakan nilai *constrain length K* = 4, input *k*=2 dan output *n*=4 maka generator matriksnya dalam bentuk oktal adalah (12, 5, 16 11). Parameter blok dari *encoder* dan *decoder* yaitu *Trellis stucture* diset menjadi *poly2trellis(4, [12 5 16 11]).*

4.2.2.1 Percobaan dengan 4 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,0625 dan *initial seed* 79. Tabel 4.21 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.22 merupakan perbandingan bit input dengan bit output.

Tabel 4.21 Perbandingan bit yang di-*encode* dengan menambahkan 4 bit error pada *K*=4, *k*=2, *n*=4

No	bit yang telah di-*encode*				bit mengalami error			
1	1	0	1	1	1	0	1	1
2	0	1	1	0	0	1	1	0
3	0	0	0	1	0	0	0	1
4	1	0	0	0	1	0	0	0
5	1	1	0	0	1	1	0	0
6	0	1	0	0	0	**0**	0	0
7	1	0	0	0	1	0	0	0
8	0	1	1	1	0	1	1	1
9	1	0	0	1	1	0	0	1
10	1	1	1	1	1	1	**0**	1
11	0	1	0	1	0	1	0	1
12	1	0	1	1	1	0	1	1
13	1	1	0	1	1	1	0	1
14	1	1	0	0	1	1	0	0
15	0	1	0	0	0	1	0	0
16	1	0	0	0	1	**1**	0	**1**

Tabel 4.22 Perbandingan bit input dengan bit output dengan *K*=4, *k*=2, *n*=4 dan 4 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 0/16

= 0

4.2.2.2 Percobaan dengan 20 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,3125 dan *initial seed* 433. Tabel 4.23 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.24 merupakan perbandingan bit input dengan bit output.

Tabel 4.23 Perbandingan bit yang di-*encode* dengan 20 bit yang mengalami error pada *K*=4, *k*=2, *n*=4

No	bit yang telah di-*encode*				bit mengalami error			
1	1	0	1	1	**0**	**1**	1	1
2	0	1	1	0	0	**0**	**0**	**1**
3	0	0	0	1	0	0	**1**	1
4	1	0	0	0	1	0	**1**	0
5	1	1	0	0	1	**0**	0	0
6	0	1	0	0	0	1	0	0
7	1	0	0	0	1	**1**	0	1
8	0	1	1	1	0	1	1	1
9	1	0	0	1	1	**1**	0	1
10	1	1	1	1	1	**0**	1	1
11	0	1	0	1	0	1	**1**	**0**
12	1	0	1	1	1	0	1	1
13	1	1	0	1	**0**	1	0	1
14	1	1	0	0	**0**	**0**	0	0
15	0	1	0	0	**1**	1	**1**	**1**
16	1	0	0	0	**0**	0	0	0

Tabel 4.24 Perbandingan bit input dengan bit output dengan *K*=4, *k*=2, *n*=4 dan 20 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	**1**	1	1	0	1	1	1	0	0	0	1	**0**	0	1	1

Probability of error = banyaknya bit yang error = 2/16
banyaknya bit input
= 0,125

4.2.2.3 Percobaan dengan 31 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,47 dan *initial seed* 271. Tabel 4.25 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.26 merupakan perbandingan bit input dengan bit output.

Tabel 4.25 Perbandingan bit yang di-*encode* dengan menambahkan 31 bit error pada *K*=4, *k*=2, *n*=4

No	bit yang telah di-*encode*				bit mengalami error			
1	1	0	1	1	1	**1**	1	**0**
2	0	1	1	0	**1**	1	**0**	0
3	0	0	0	1	**1**	0	**1**	1
4	1	0	0	0	1	0	0	0
5	1	1	0	0	1	**0**	**1**	1
6	0	1	0	0	0	**0**	**1**	0
7	1	0	0	0	**0**	0	**1**	0
8	0	1	1	1	**1**	**0**	1	1
9	1	0	0	1	1	**1**	**1**	1
10	1	1	1	1	1	**0**	1	**0**
11	0	1	0	1	**1**	1	0	**0**
12	1	0	1	1	**0**	**1**	**0**	**0**
13	1	1	0	1	1	**0**	0	1
14	1	1	0	0	1	**0**	0	**1**
15	0	1	0	0	0	**0**	0	0
16	1	0	0	0	**0**	0	**1**	**1**

Tabel 4.26 Perbandingan bit input dengan bit output dengan *K*=4, *k*=2, *n*=4 dan 30 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	**1**	1	**0**	**1**	1	1	**0**	0	**1**	**1**	1	**0**	0	**0**	1

Probability of error = banyaknya bit yang error = 8/16
banyaknya bit input
= 0,5

4.3 Percobaan dengan *Constrain Length K*=7

4.3.1 Percobaan dengan Input *k*=1 dan Output *n*=2

Pada percobaan ini menggunakan nilai *constrain length K* = 7, input *k*=1 dan output *n*=2 maka generator matriksnya dalam bentuk oktal adalah (171, 133) seperti yang ditunjukkan pada Tabel 4.1 di atas. Parameter blok dari *encoder* dan *decoder* yaitu *Trellis stucture* diset menjadi *poly2trellis(3, [7 5])*. Dalam setiap

satu bit input *encoder* akan menghasilkan dua bit *redundant* sehingga terdapat tiga bit pada output dari *encoder*. Percobaan ini menggunakan data input biner yang berasal dari *Bernoulli binary random generator* sebanyak 16 bit atau 2 Byte sehingga dengan spesifikasi *encoder* maka input dengan 16 bit akan di*encode* menjadi dua kalinya yaitu 32 bit yang berisi 16 bit informasi asli dan 16 bit *redundant*.

4.3.1.1 Percobaan dengan 4 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,175 dan *initial seed* 88. Tabel 4.27 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.28 merupakan perbandingan bit input dengan bit output.

Tabel 4.27 Perbandingan bit yang di-*encode* dengan 4 bit yang mengalami error pada *K*=7, *k*=1, *n*=2

No	bit yang telah di-*encode*		bit mengalami error	
1	1	1	1	1
2	1	0	**0**	0
3	0	0	0	0
4	1	0	**0**	**1**
5	0	1	0	1
6	1	0	1	0
7	0	1	0	1
8	1	1	1	**0**
9	0	0	0	0
10	1	1	1	1
11	1	0	1	0
12	0	1	0	1
13	1	1	1	1
14	1	0	1	0
15	1	1	1	1
16	1	0	1	0

Tabel 4.28 Perbandingan bit input dengan bit output dengan *K*=7, *k*=1, *n*=3 dan 4 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	1	0	1	**0**	1	**1**	**1**	0	1	1	0	1	1

Probability of error = banyaknya bit yang error = 3/16
banyaknya bit input
= 0,1875

4.3.1.2 Percobaan dengan 10 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,3125 dan *initial seed* 62. Tabel 4.29 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.30 merupakan perbandingan bit input dengan bit output.

Tabel 4.29 Perbandingan bit yang di-*encode* dengan menambahkan 10 bit error pada *K*=7, *k*=1, *n*=2

No	bit yang telah di-*encode*		bit mengalami error	
1	1	1	1	**0**
2	1	0	**0**	0
3	0	0	0	0
4	1	0	1	**1**
5	0	1	0	1
6	1	0	1	0
7	0	1	0	**0**
8	1	1	1	1
9	0	0	**1**	0
10	1	1	**0**	1
11	1	0	1	0
12	0	1	0	**0**
13	1	1	**0**	1
14	1	0	1	**1**
15	1	1	**0**	1
16	1	0	1	0

Tabel 4.30 Perbandingan bit input dengan bit output dengan *K*=7, *k*=1, *n*=2 dan 10 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	**0**	0	**0**	1	**1**	1	**0**	1	0	0	**1**	1	1	0	**0**	1

Probability of error = banyaknya bit yang error = 6/16
banyaknya bit input
= 0,375

4.3.1.3 Percobaan dengan 18 bit error pda urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,25 dan *initial seed* = 33. Tabel 4.31 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.32 merupakan perbandingan bit input dengan bit output.

Tabel 4.31 Perbandingan bit yang di-*encode* dengan menambahkan 18 bit error pada *K*=7, *k*=1, *n*=2

No	bit yang telah di-*encode*		bit mengalami error	
1	1	1	1	**0**
2	1	0	1	**1**
3	0	0	0	0
4	1	0	**0**	0
5	0	1	**1**	1
6	1	0	1	0
7	0	1	**1**	1
8	1	1	**0**	**0**
9	0	0	**1**	**1**
10	1	1	1	**0**
11	1	0	**0**	**1**
12	0	1	0	1
13	1	1	**0**	**0**
14	1	0	**0**	0
15	1	1	1	**0**
16	1	0	**0**	**1**

Tabel 4.32 Perbandingan bit input dengan bit output dengan *K*=7, *k*=1, *n*=2 dan 18 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	**0**	**1**	**0**	1	**1**	1	**0**	1	0	0	**1**	1	**0**	**1**	**0**	1

Probability of error = banyaknya bit yang error = 9/16
banyaknya bit input
= 0,5625

4.3.2 Percobaan Dengan Input *k*=1 dan Output *n*=3

Pada percobaan ini menggunakan nilai *constrain length K* = 3, input *k*=1 dan output *n*=3 maka generator matriksnya dalam bentuk oktal adalah (171, 133, 122). Parameter blok dari *encoder* dan *decoder* yaitu *Trellis stucture* diset menjadi *poly2trellis(3, [171 133 122]).*

4.3.2.1 Percobaan dengan 6 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,16 dan *initial seed* = 47. Tabel 4.33 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.34 merupakan perbandingan bit input dengan bit output.

Tabel 4.33 Perbandingan bit yang di-*encode* dengan menambahkan 6 bit error pada *K*=7, *k*=1, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	1	1
2	1	0	0	1	0	0
3	0	0	0	0	0	0
4	1	0	1	1	0	1
5	0	1	1	0	1	1
6	1	0	1	1	0	1
7	0	1	1	0	1	1
8	1	1	1	1	1	1
9	0	0	0	**1**	0	0
10	1	1	1	1	1	**0**
11	1	0	1	1	0	1
12	0	1	0	0	**0**	0
13	1	1	0	1	1	0
14	1	0	1	1	0	**0**
15	1	1	0	1	**0**	**1**
16	1	0	1	1	0	1

Tabel 4.34 Perbandingan bit input dengan bit output dengan *K*=7, *k*=1, *n*=3 dan 6 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 0/16

= 0

4.3.2.2 Percobaan dengan 13 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,16 dan *initial seed* = 47. Tabel 4.35 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.36 merupakan perbandingan bit input dengan bit output.

Tabel 4.35 Perbandingan bit yang di-*encode* dengan menambahkan 13 bit error pada *K*=7, *k*=1, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	1	**0**
2	1	0	0	1	**1**	0
3	0	0	0	0	0	0
4	1	0	1	1	**1**	**0**
5	0	1	1	0	**0**	1
6	1	0	1	1	**1**	1
7	0	1	1	**1**	**0**	1
8	1	1	1	1	1	1
9	0	0	0	0	0	0
10	1	1	1	**0**	**0**	**0**
11	1	0	1	1	0	1
12	0	1	0	0	1	0
13	1	1	0	1	1	**1**
14	1	0	1	1	0	1
15	1	1	0	1	1	**1**
16	1	0	1	1	0	1

Tabel 4.36 Perbandingan bit input dengan bit output dengan *K*=7, *k*=1, *n*=3 dan 13 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	**0**	**1**	1	**0**	1	0	**1**	**1**	**0**	1	**1**	**0**	1

Probability of error = banyaknya bit yang error / banyaknya bit input = 8/16 = 0,5

4.3.2.3 Percobaan dengan 24 bit error pada urutan yang di-encode

Parameter blok dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,475 dan *initial seed* = 19. Tabel 4.37 berikut ini merupakan perbandingan antar hasil urutan pesan yang di-*encode* dengan urutan pesan yang mengalami error dan Tabel 4.38 merupakan perbandingan bit input dengan bit output.

Tabel 4.37 Perbandingan bit yang di-*encode* dengan menambahkan 24 bit error pada *K*=7, *k*=1, *n*=3

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	**0**	**0**	**0**
2	1	0	0	1	**1**	**1**
3	0	0	0	0	**1**	**1**
4	1	0	1	1	**1**	**0**
5	0	1	1	0	1	1
6	1	0	1	**0**	0	**0**
7	0	1	1	0	1	**0**
8	1	1	1	1	1	1
9	0	0	0	0	0	**1**
10	1	1	1	**0**	**0**	1
11	1	0	1	**0**	**1**	1
12	0	1	0	**1**	1	0
13	1	1	0	**0**	1	0
14	1	0	1	1	0	**0**
15	1	1	0	**0**	1	0
16	1	0	1	**0**	**1**	**0**

Tabel 4.38 Perbandingan bit input dengan bit output dengan *K*=7, *k*=1, *n*=3 dan 24 bit error pada bit yang di-*encode*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	**0**	**1**	1	1	**1**	**0**	1	1	**1**	**1**	**1**	1	**0**	**1**	1	1

Probability of error = banyaknya bit yang error = 9/16
banyaknya bit input
= 0,5625

Tabel 4.39 dibawah ini merupakan merupakan rangkuman dari tabel-tabel di atas yaitu perbandingan *probability of error* dengan berbagai parameter nilai *constrain length K*, *k*, dan *n*, serta jumlah bit error yang digunakan

Tabel 4.39 Perbandingan Probability of error dengan berbagai parameter *K*, *k*, dan *n*, serta jumlah bit error

Constrain length	parameter convolutional code (k,n)	bit error pada saluran transmisi	probability of error
K=3	(1,2)	5 bit	0.25
		10 bit	0.5
		18 bit	0.5
	(1,3)	6 bit	0.0625
		12 bit	0,1875
		26 bit	0,3125
K=4	(2,3)	4 bit	0
		10 bit	0,25
		21 bit	0,4375
	(2,4)	4 bit	0
		20 bit	0,125
		31 bit	0,5
K= 7	(1,2)	4 bit	0,1875
		10 bit	0,375
		18 bit	0,5625
	(1,3)	6 bit	0
		13 bit	0,5
		24 bit	0,5625

Dari data-data tabel di atas terlihat proses *encoding* yang menambahkan bit input dengan bit *redundant* yang berfungsi untuk mendeteksi dan mengkoreksi error oleh *decoder*. Bit error yang muncul pada urutan yang di-*decode* terletak pada bit *redundant* dan juga bit informasi. Dengan adanya bit *redundant* ini *decoder* dapat mendeteksi dan mengkoreksi error yang terjadi namun dapat terlihat pada hasil percobaan bahwa bit informasi dengan berbagai nilai parameter dari *encoder* dan *decoder* tersebut tidak dapat diperbaiki semua sehingga menimbulkan suatu nilai yaitu *probability of error*.

Berdasarkan Tabel 4.39 di atas, dapat dilihat bahwa untuk rate yang lebih besar akan menyebabkan nilai error semakin rendah. Hal ini terjadi karena encoder menghasilkan sinyal redundant yang lebih banya dari sinyal informasi yang masuk sehingga kemungkinan besar error akan menyerang bit *redundant*. Jadi dengan jumlah bit input yang masuk per waktu *k* yang kecil dan jumlah bit output yang keluar *n* yang besar akan lebih mempertahankan keaslian dari sinyal

yang dikirim sehingga *decoder* lebih efektif dalam mendeteksi dan mengkoreksi error yang terjadi. *Probability of error* juga berpengaruh terhadap nilai *constrain length*, dari tabel di atas terlihat bahwa dengan *constrain length K* yang lebih besar maka *Probability of error* akan semakin rendah. Hal ini terjadi karena *constrain length* juga menentukan proses *encoding* suatu sinyal yaitu dalam melakukan penambahan sinyal *redundant* pada sinyal informasi dengan metode *shift register* dan *modulo 2 adder*. Tetapi dengan penambahan *constrain length K* yang semakin banyak, akan menyebabkan biaya produksi rangkaian *convolutional encoder* dan *Viterbi decoder* semakin mahal karena banyaknya *shift register* yang digunakan.

4.4 Uji Coba pada DSK TMS320C6713

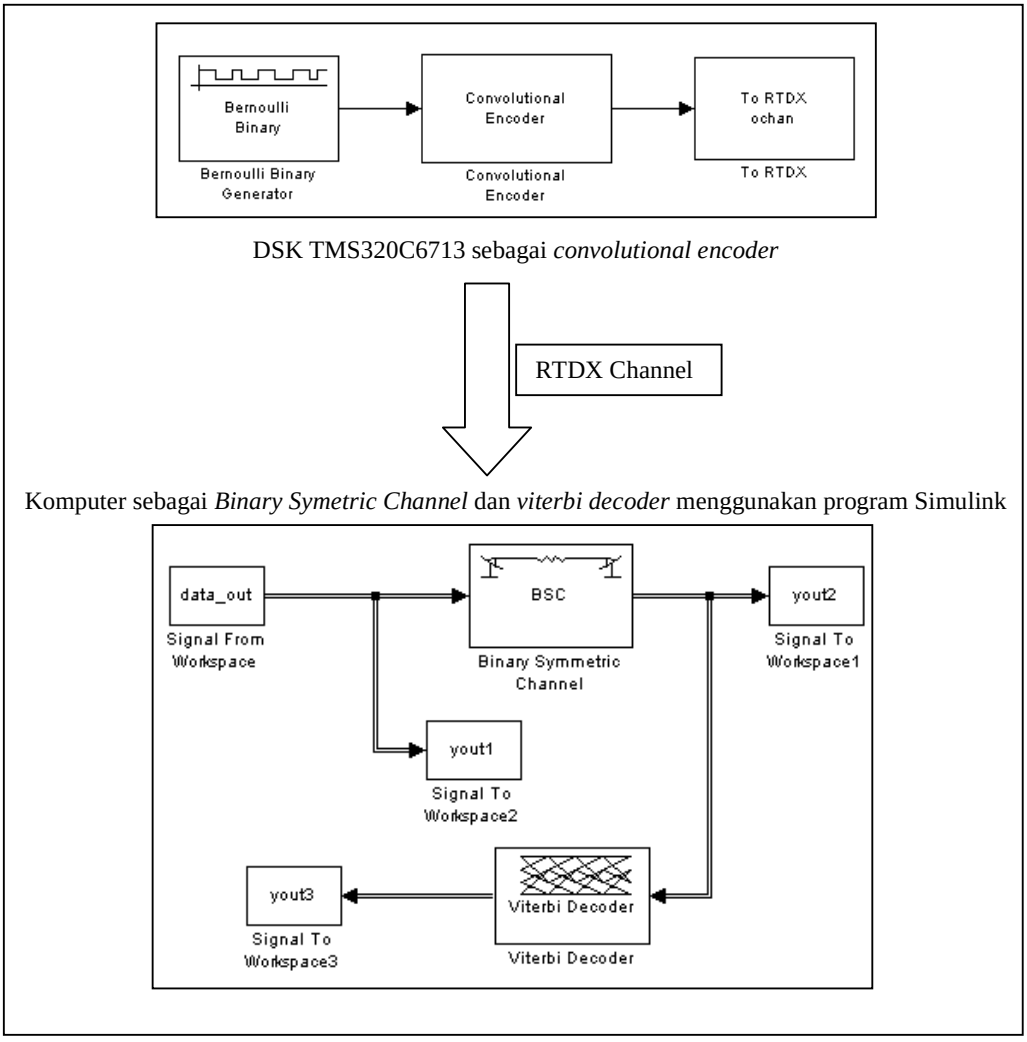
Uji coba yang dilakukan dengan mengimplementasikan program dari rangkaian *convolutional encoder* dan *viterbi decoder* yang dibuat dalam bentuk model pada program Simulink ke dalam DSK TMS320C6713. Skematik rangkaian ini adalah antara DSK TMS320C6713 sebagai *convolutional* encoder atau *viterbi decoder* dengan komputer untuk melihat hasil dari proses *encoding* dan *decoding*.

4.4.1 Percobaan Dengan DSK TMS320C6713 Sebagai *Convolutional Encoder*

Pada percobaan ini *convolutional encoder* terletak pada DSK TMS320C6713 sedangkan *Binary Symetric Channel* dan *viterbi decoder* terletak pada komputer dengan program Simulink. Uji coba dilakukan dengan data input biner yang berasal dari *Bernoulli Binary Random Generator* yang diprogram ke dalam DSK tersebut dan hasil keluaran dari *convolutional encoder* dapat ditampilkan melalui *RTDX Channel* dan menyimpannya ke dalam array pada MATLAB. *RTDX Channel* dapat dijalankan dengan membuat *file* atau *driver* dalam bentuk (.m) yang berisi perintah (*syntax*) untuk menjalankan saluran *RTDX*.

Hasil *encoding* yang disimpan dalam array dikeluarkan dengan menggunakan blok *workspace* untuk diteruskan ke *Binary Symetric Channel* (BSC) lalu masuk ke dalam *viterbi decoder* yang kemudian hasilnya merupakan

hasil output dari rangkaian keseluruhan. Gambar 4.1 di bawah ini menujukkan alur dari uji coba yang dilakukan.



Gambar 4.1 Alur dari uji coba dengan DSK TMS320C6713 sebagai *convolutional encoder*

Uji coba ini menggunakan parameter *constrain length K* = 7, *k* = 1, dan *n* = 2. Parameter blok dari *encoder dan decoder* adalah *Polytrellis(7, [171 133])* dan parameter *Binary Symetric Channel* (BSC) dengan *error probability* = 0,175 yang akan menambahkan 4 bit error ke dalam bit yang sudah di-*encode* dan *initial seed* diset 88. Tabel 4.40 di bawah ini merupakan bit input yang dihasilkan oleh *Bernoulli Binary Random Generator* yang terletak pada DSK TMS320C6713 dan Tabel 4.41 merupakan perbandingan urutan pesan yang di-*encode* dengan penambahan bit error

Tabel 4.40 Bit input rangkaian *convolutional encoder* pada DSK TMS320C6713

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1

Tabel 4.41 Perbandingan urutan bit yang di-*encode* dengan penambahan 4 bit error

No	bit yang telah di-*encode*		bit mengalami error	
1	1	1	1	1
2	1	0	**0**	0
3	0	0	0	0
4	1	0	**0**	**1**
5	0	1	0	1
6	1	0	1	0
7	0	1	0	1
8	1	1	1	**0**
9	0	0	0	0
10	1	1	1	1
11	1	0	1	0
12	0	1	0	1
13	1	1	1	1
14	1	0	1	0
15	1	1	1	1
16	1	0	1	0

Tabel 4.42 Perbandingan bit input dan output pada DSK TMS320C6713 sebagai *convolutional encoder*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	1	0	1	**0**	1	**1**	**1**	0	1	1	0	1	1

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}}$ = 3/16

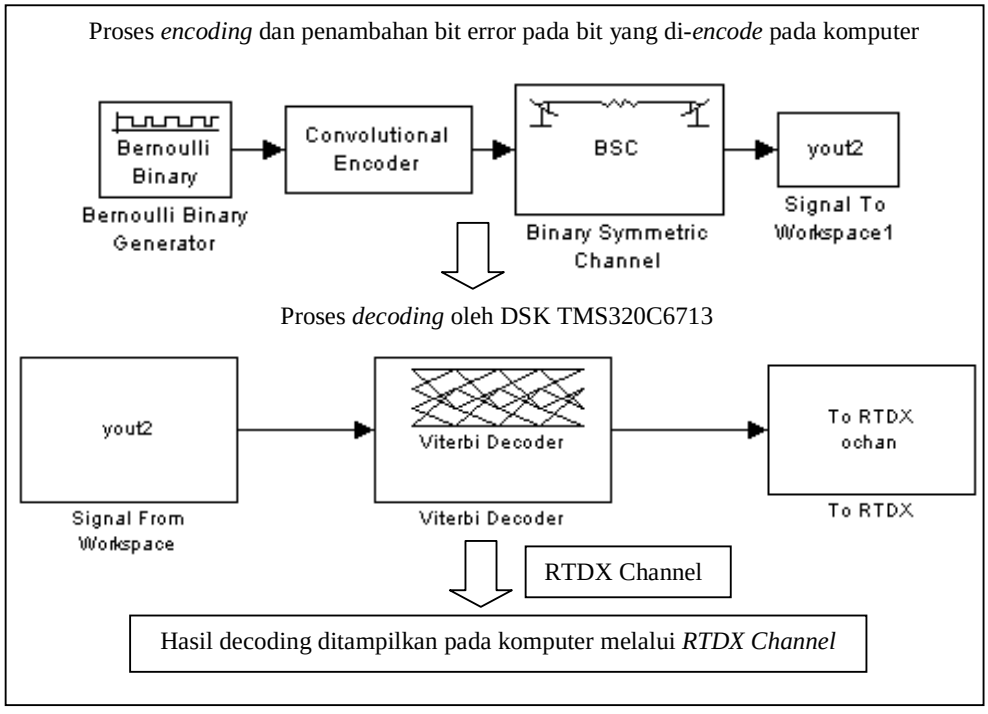
= 0,1875

Tabel di atas menunjukkan hasil output yang memiliki error 3 bit sehingga *probability of error* = 0,1875. Uji coba rangkaian dengan DSK TMS320C6713 sebagai *convolutional encoder* dengan simulasi menggunakan program Simulink dengan nilai K = 7, k = 1, dan n = 2 menunjukkan hasil yang sama. Hal ini

menunjukkan bahwa DSK TMS320C6713 dapat dirancang menjadi *convolutional encoder* dan memiliki kemampuan untuk meng-*encode* dengan metode shift register sesuai dengan teori yang telah dijelaskan sebelumnya.

4.4.2 Percobaan Dengan DSK TMS320C6713 Sebagai *Viterbi Decoder*

Pada percobaan ini komputer dengan menggunakan program Simulink akan digunakan sebagai *convolutional encoder* dan DSK TMS320C6713 sebagai *viterbi decoder*. Input *convolutional encoder* berasal dari blok *Bernoulli Binary Random Generator* dan ouputnya akan ditambahkan dengan error menggunakan *Binary Symetric Channel* dan kemudian hasilnya akan di-*decode* oleh DSK TMS320C6713. Input pada DSK TMS320C6713 berasal dari blok *signal to workspace*, blok ini berisi bit yang sudah di-*encode* yang berasal dari hasil simulasi pada program simulink. Hasil dari *decoding* yang dilakukan oleh DSK akan ditampilkan ke dalam komputer melalui *RTDX Channel.* Gambar 4.2 di bawah ini merupakan alur uji coba dengan DSK sebagai *viterbi decoder*.



Gambar 4.2 Alur uji coba DSK TMS320C6713 sebagai *viterbi decoder*

Pada percobaan ini menggunakan nilai *constrain length* K = 7, input k=1 dan output n=3 maka generator matriksnya dalam bentuk oktal adalah (171, 133, 122). Parameter blok dari *encoder* dan *decoder* yaitu *Trellis stucture* diset menjadi *poly2trellis(7, [171 133 122])*. Parameter dari *Binary Symetric Channel* (BSC) adalah *error probability* = 0,16 yang akan menambahkan delapan bit error ke dalam bit yang sudah di-*encode* dan *initial seed* diset 47. Tabel 4.43 di bawah ini merupakan bit input yang dihasilkan oleh *Bernoulli Binary Random Generator* dan Tabel 4.44 merupakan perbandingan urutan pesan yang di-*encode* dengan penambahan 6 bit error.

Tabel 4.43 Bit input rangkaian *convolutional encoder* pada DSK TMS320C6713

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1

Tabel 4.44 Perbandingan urutan bit yang di-*encode* dengan penambahan 6 bit error

No	bit yang telah di-*encode*			bit mengalami error		
1	1	1	1	1	1	1
2	1	0	0	1	0	0
3	0	0	0	0	0	0
4	1	0	1	1	0	1
5	0	1	1	0	1	1
6	1	0	1	1	0	1
7	0	1	1	0	1	1
8	1	1	1	1	1	1
9	0	0	0	**1**	0	0
10	1	1	1	1	1	**0**
11	1	0	1	1	0	1
12	0	1	0	0	**0**	0
13	1	1	0	1	1	0
14	1	0	1	1	0	**0**
15	1	1	0	1	**0**	**1**
16	1	0	1	1	0	1

Tabel 4.45 Perbandingan bit input dan output pada DSK TMS320C6713 sebagai *viterbi decoder*

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit input	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1
Bit output	1	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1

Probability of error = $\frac{\text{banyaknya bit yang error}}{\text{banyaknya bit input}} = 0/16$

= 0

Dari hasil uji coba terlihat bahwa bit input yang di-*encode* kemudian diberi error sebesar delapan bit oleh *Binary Symetric Channel* pada komputer dan kemudian di-*decode* oleh DSK TMS320C6713 menghasilkan tidak ada error pada output. Hal serupa juga terjadi pada simulasi menggunakan Simulink yang dilakukan pada subbab sebelumnya. Hal ini menunjukkan bahwa *viterbi decoder* dapat dirancang ke dalam DSK TMS320C6713 dengan mendeteksi dan mengkoreksi error pada bit informasi yang asli menggunakan sinyal *redundant*.

BAB 5
KESIMPULAN

Berdasarkan hasil uji coba dan analisis yang telah dilakukan pada bab sebelumnya, maka dapat disimpulkan bahwa rangkaian *convolutional encoder* dan *viterbi decoder* dapat diimplementasikan ke dalam DSK TMS320C6713 dengan bantuan program Simulink dan juga dengan berbagai kondisi yaitu :

1. Semakin kecil *rate* atau perbandingan antara bit input yang masuk perwaktu dengan bit output yang dihasilkan, maka *probability of error* akan semakin kecil karena jumlah bit sinyal redundant yang dihasilkan encoder lebih banyak dari pada jumlah bit input. Sehingga menyebabkan *error* kemungkinan besar terletak pada bit *redundant*.
2. *Constrain length K* juga berpengaruh terhadap nilai *probability of error*. Semakin besar *K* yang digunakan maka *probability of error* akan semakin kecil, tetapi menyebabkan biaya produksi rangkaian *convolutional encoder* dan *Viterbi decoder* semakin mahal.
3. *Convolutional encoder dan viterbi decoder* yang disimulasikan dengan program simulink dan juga diuji coba pada DSK TMS320C6713 tidak dapat memperbaiki semua error.

DAFTAR ACUAN

[1] K. Sham Shanmugam, *Digital and Analog Communication System* (New York : John Wiley & Sons Inc, 1979), Hal. 443-448, 478-480

[2] John G. Proakis, *Digital Communications* Third Edition (Singapore: McGraw-Hill, 1995), Hal. 470-477

[3] Muhammad Suryanegara, "*Analisa Konfigurasi Serial Concatenated Code Dengan Menggunakan Teknik Forward Error Correction (FEC) Convolutional Code dan A Posteriory Probability (APP) Decoder"*, Skripsi, Departemen Elektro Fakultas Teknik Universitas Indonesia, Jakarta, 1999, Hal. 6-8

[4] Shu Lin, Daniel J. Costello, *Error Control Coding: Fundamentals and Applications* (Englewood Cliffs, New Jersey: Prentice-Hall, 1983), Hal. 315

[5] Jorge Castineira Moreira & Patrick Guy Farrel, *Essential of Error Control Coding* (England: John Wiley & Sons, Ltd, 2006), Hal. 182

[6] Rulph Chassaing, *Digital Signal Processing and Application with the C6713 and C6416 DSK* (New Jersey: John Wiley & Sons, Inc., 2005), Hal. 404-414

DAFTAR PUSTAKA

Blahut, Richard E. (1984). *Theory and Practice of Error Control Codes*. Addison-Wesley Publishing Company.

Lin, Shu, & Costello, Jr., Daniel J. (1983). *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, New Jersey: Prentice-Hall.

MATLAB HELP

Wilson, Stephen G. (1996). *Digital Modulation and Coding*. New Jersey: Prentice-Hall International, Inc.

DAFTAR LAMPIRAN

Lampiran 1 Listing program driver “To RTDX”.

```
function RTDXdriver(modelname)
% RTDXDRIVER Reads and plots data from an RTDX channel

[modelpath,modelname,modelext] = fileparts(modelname);

cc = ccsdsp;
set(cc,'timeout',50);
if ~isrtdxcapable(cc)
   error('Processor does not RTDX support');
end

cc.reset; pause(1);
cc.cd(modelpath);
cc.visible(1);

open(cc,sprintf('%s.pjt',modelname));
load(cc,sprintf('%s.out',modelname));

rx = cc.rtdx;
rx.set('timeout', 100); % Reset timeout = 10 seconds

rx.configure(64000,2);

rx.open('ochan','r');

rx.enable; % enable RTDX

cc.run; % cc.enable can be placed here

pause(1); % cc.enable cannot be placed here; too much time had passed
       % RTDX processing will be 'stalled'
if isenabled(rx,'ochan')
   for a=1:1001
data_out((2*a-1):(2*a),1)=readmsg(cc.rtdx,'ochan','uint8',[2 1],1);
   end
   end
save('hasil','data_out');
%pause(40);
RTDXcleanup(cc,rx);
%=========================================================================
% Put RTDX back to good state
%=========================================================================
function RTDXcleanup(cc,rx)
if isrunning(cc),        % if the target DSP is running
   halt(cc);           % halt the processor
end

cc.reset;
disable(rx,'ochan');
disable(rx);            % disable RTDX
close(cc.rtdx,'ochan');
```