DESAIN DAN ANALISIS PENDEKODE VITERBI MENGGUNAKAN SATU *BUTTERFLY* BERBASIS BAHASA VHDL

Iswahyudi Hidayat
Departemen Teknik Elektro - Institut Teknologi Telkom Bandung
e-mail: isw@stttelkom.ac.id

Abstraks
Pengkode konvolusi telah digunakan secara luas pada sistem komunikasi, khususnya pada sistem komunikasi wireless yang berkembang saat ini. Pada bagian pengirim, pengkode konvolusi bekerja dengan membangkitkan keluaran data stream dari bit-bit masukan. Sedangkan penerima akan melakukan proses pendekode untuk memperoleh kembali bit informasi asli sebelum melewati pengkode konvolusi. Algoritma yang banyak diusulkan penggunaannya pada bagian penerima adalah algoritma viterbi.
Penelitian ini membahas tentang desain pendekode viterbi menggunakan bahasa pemrograman VHDL (Very High Speed IC Description Language). Desain yang diusulkan menggunakan sebuah butterfly pada bagian penerima. Pada umumnya butterfly yang digunakan merupakan fungsi persamaan 2^{K-1}. Dengan K adalah parameter Constrain Length. Di sisi lain fungsi dari setiap butterfly dalam proses pendekode viterbi adalah sama. Sehingga pada penelitian ini akan coba didesain penerima viterbi dengan sebuah blok butterfly yang digunakan secara bergantian. Nilai constrain length (K) yang digunakan adalah K = 7 dan code rate = ½ (satu per dua). Dengan K = 7 lazimnya jumlah butterfly yang diperlukan adalah 32.
Hasil desain akhir dari penelitian ini adalah sintesis dan layout rangkaian pendekode viterbi. Hasil sintesis rangkaian berupa area sintesis dan frekuensi kerja yang dihasilkan. Proses layout hasil desain menggunakan standard teknologi CMOS 0,18 mikron. Pada proses layout, hasil penelitian yang ditunjukkan berupa frekuensi kerja, kecepatan data, ukuran core, dan ukuran chip.

***Kata Kunci** : Pengkode Konvolusi, Pendekode Viterbi, Constrain Length, Butterfly, sintesis, layout.*

1. PENDAHULUAN

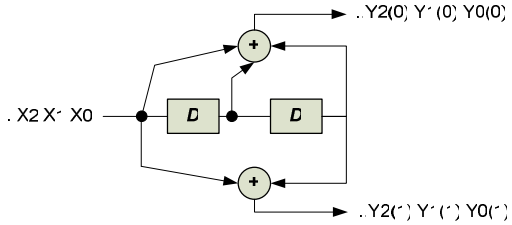
Algoritma pendekode viterbi pertama kali diperkenalkan oleh Andrew J. Viterbi pada tahun 1967. Algoritma ini digunakan dalam proses pendekode (*decoder*) bagi pengkode konvolusi. Konsep utama dari pendekode viterbi ini adalah menentukan jalur dengan jarak Hamming terkecil dari sejumlah jalur yang mungkin dilalui pada diagram trellis proses pengkode.
Desain sistem elektronika yang berkembang dewasa ini seluruhnya berbasis *software* disain.
Dalam penelitian ini akan dilakukan desain pengkode konvolusi dan pendekode viterbi. Disain diawali dari pengujian fungsional sistem. Jika pengujian fungsional sistem telah terselesaikan, maka dilanjutkan pada sintesis rangkaian hasil perancangan. Sintesis dilakukan dengan menggunakan design analyzer dengan class library *synthesis* dari synopsys *synthesis tools*. Hasil sintesis yang diperoleh selanjutnya akan digunakan sebagai masukan proses layout. Layout terdiri dari tahapan floorplanning dan routing dilakukan pada teknologi CMOS 0,18 mikron.

2. TINJAUAN PUSTAKA

2.1 Algoritma Pendekode Viterbi

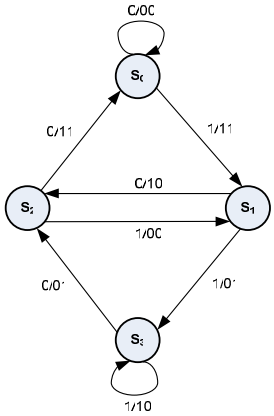
2.1.1 Pengkode Konvolusi

Pengkodean konvolusi telah digunakan secara luas pada sistem komunikasi terutama pada komunikasi wireless yang saat ini berkembang pesat. Pengkode konvolusi dapat dipergunakan pada rentetan masukan yang kontinu (yang tidak dapat dilakukan oleh pengkodean blok). Dalam kenyataannya pengkodean konvolusi dapat dipandang sebagai *finite state machine* (FSM). Pengkode konvolusi membangkitkan keluaran data stream terkodekan dari suatu masukan data stream. Pengkode konvolusi ini terdiri dari sejumlah shift register dan fungsi persamaan gerbang XOR (*Exclusive OR*) seperti contoh berikut ini :



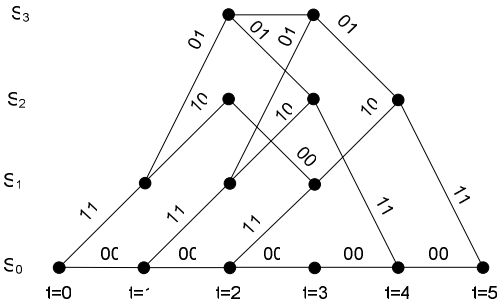
Gambar 1 Pembangkitan Pengkode Konvolusi

Operasi pengkode konvolusi akan lebih mudah ditunjukkan dengan diagram *state*. Gambar 2 menunjukkan diagram *state* pengkode yang ditunjukkan pada gambar 1. Gambar 2 menggambarkan transisi *state* dan keluaran pengkode yang dihasilkan pada setiap transisi *state*.



Gambar 2 Diagram *State* Pengkode Konvolusi Gambar 1

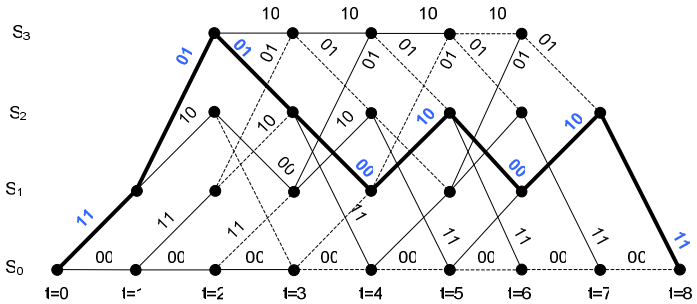
Diagram trellis adalah cara lain menunjukkan transisi *state* pada pengkode konvolusi beserta keluaran bit pengkode yang dihasilkan dengan kelebihan cara ini lebih mudah melihat transisi dalam domain waktunya. Gambar 3 menunjukkan diagram trellis pada pengkode konvolusi gambar 1. Diagram trellis yang ditunjukkan pada gambar 3 adalah untuk jumlah bit masukan sebanyak 5 bit, dengan 2 bit terakhir menunjukkan proses reset menuju *state* awal.



Gambar 3 Diagram Trellis Pengkode Gambar 1

2.1.2 Konsep Algoritma Pendekode Viterbi

Algoritma viterbi yang diimplementasikan dengan persyaratan algoritma ML (*Maximum Likelihood*) dan secara *hard decision* ditunjukkan pada gambar 4. Pengkode konvolusi akan mengkodekan informasi asli dengan urutan (11010100) dan menghasilkan keluaran pengkode dengan *code rate* 1/2 dengan urutan (11,01,01,00,10,00,10,11). Hasil pengkode ini selanjutnya dikirimkan pada kanal transmisi yang tercampur derau. Dan pada bagian penerima dimisalkan diperoleh urutan kode (10,01,01,01,10,00,10,11). Dari perbandingan kode keluaran pengkode (*encoder*) konvolusi dan kode terima pada pendekode (*decoder*) viterbi, terlihat adanya perbedaan yang ditimbulkan oleh derau pada kanal.



Gambar 4 Diagram Trellis Pada *Hard Decision Decoding*

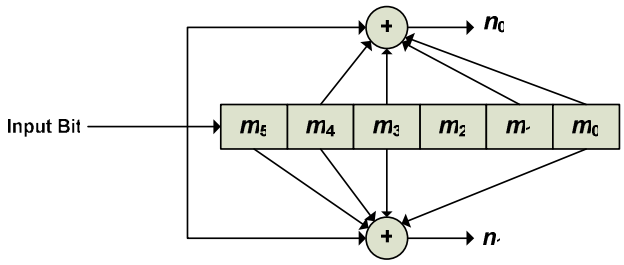
Prosedur *traceback* pada pendekode (*decoder*) terlihat pada diagram trelllis pada gambar 4. Proses *traceback* ini berawal dari konsep bahwa setiap percabangan (branch) terkait dengan bit masukan tertentu pada pengkode. Sebagai contoh, percabangan dari *state* S_2 pada t = 7 menuju *state* S_0 pada t = 8 berhubungan dengan masukan bit '0' pada pengkode.

3. METODE PENELITIAN

3.1 Perancangan Pengkode Konvolusi dan Pendekode Viterbi

3.1.1 Bagian Pengkode Konvolusi

Pada bagian ini, informasi asli akan diproses untuk menghasilkan suatu format pengkodean konvolusi. Dengan *constrain length* = 7 dan laju kode (*code rate*) = ½, maka diperoleh format pembangkitan kode sebagai berikut :



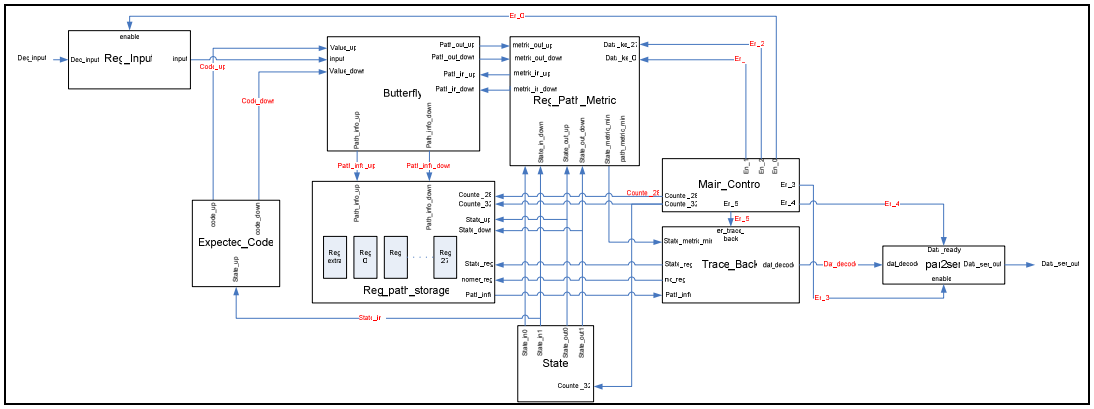
Gambar 5 Blok Diagram Pengkode Konvolusi

Pada desain bagian pengkode konvolusi ini, *generator polynomial* untuk pembangkitan n_0 dan n_1 memiliki bentuk sebagai berikut :

$$n_0 = 133_o = 1\ 011\ 011_2 \quad (1)$$
$$n_1 = 171_o = 1\ 111\ 001_2 \quad (2)$$

3.1.2 Bagian Pendekode

Proses pendekodean pengkode konvolusi menggunakan algoritma viterbi. Konsep dasar algoritma viterbi dijelaskan pada bagian landasan teori pada bab II. Pada bab ini, algoritma viterbi telah diterjemahkan dalam bentuk arsitektur yang diusulkan yang selanjutnya akan diimplementasikan dalam bahasa VHDL. Arsitektur umum suatu pendekode menggunakan algoritma viterbi dapat ditunjukkan sebagai berikut :



Gambar 6 Arsitektur Pendekode Viterbi dengan 1 Butterfly

Dalam penelitian ini akan coba diuji 2 arsitektur pendekode yang berbeda pada jumlah blok *butterfly*. Blok *butterfly* merupakan blok gabungan dari *branch metric* dan ACS (*Add compare select*). Arsitektur pertama yang diusulkan adalah arsitektur dengan sebuah *butterfly*, sedangkan arsitektur lainnya adalah arsitektur dengan menggunakan empat buah *butterfly*. Alasan pemilihan arsitektur pertama adalah karena fungsi blok *butterfly* yang sama untuk ketiga puluh dua jenis sel. Sehingga dengan sebuah *butterfly* dapat mewakili fungsi 32 sel yang harus diproses dengan pengaturan control parameter setiap sel yang saat itu sedang diproses oleh *butterfly*. Sedangkan bentuk arsitektur dengan empat *butterfly* dipilih karena dari ketiga puluh dua jenis sel yang ada, jika dikelompokkan akan terdapat empat jenis saja.

A. Blok Kontrol

Merupakan bagian yang mengendalikan seluruh proses yang terjadi pada pendekode. Unsur utama pada blok kontrol ini adalah rangkaian *counter* (penghitung). Pada penggunaan sebuah *butterfly*, maka dibutuhkan rangkaian *counter* 28 dan *counter* 32. *Counter* 28 merepresentasikan jumlah data yang akan diolah pada suatu interval acuan. Jumlah data ini berhubungan dengan parameter *constrain length* yang dipergunakan pada pendekode dengan menggunakan algoritma viterbi. Hubungan antara jumlah data yang diolah sebelum data selanjutnya diproses mengikuti bentuk persamaan :

$$\Sigma \text{ data proses} = 4K \quad ; \quad K = \textit{constrain length} \tag{3}$$

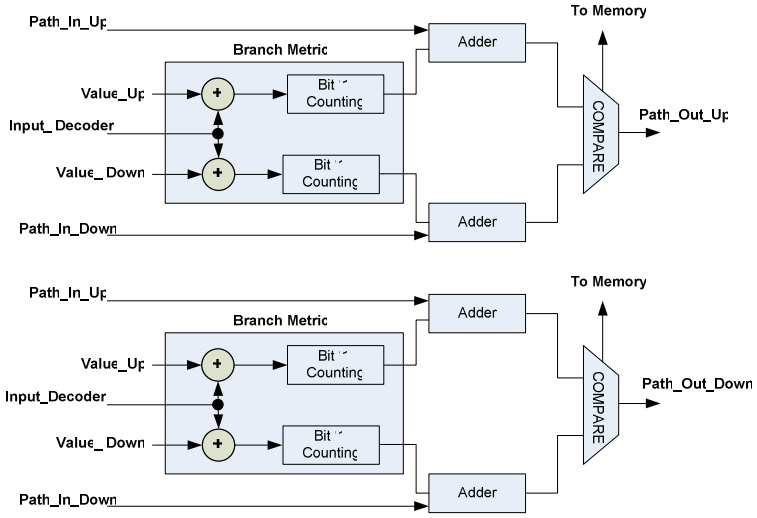
Sedangkan *counter* 32 menunjukkan jumlah kombinasi *state* yang terdapat pada pengkode trellis. Hal ini digunakan dalam mengatur laju data dengan mengaktifasi sinyal *enable.*

B. Blok Antarmuka Masukan

Berfungsi untuk membuffer data masukan selama satu pulsa *clock*, sehingga diperoleh waktu acuan sebuah data masukan dianggap valid untuk diproses. Artinya jika terjadi perubahan masukan data selama kurang dari periode satu pulsa *clock* sejak data terakhir diterima, maka perubahan tersebut akan diabaikan. Pada desain pendekode ini, register input dikontrol oleh sinyal en_0 (*enable* ke-0) yang dihasilkan oleh blok *main_control*.

C. *Butterfly*

Bagian ini berfungsi untuk melakukan pengecekan kondisi data masukan yang diterima oleh pendekode dan akan dibandingkan dengan nilai data yang seharusnya diterima. Pembandingan ini akan dinyatakan dalam penghitungan jarak hamming yang terjadi. Selain melakukan penghitungan jarak hamming, pada bagian ini juga akan melakukan seleksi terhadap jalur *state* – jalur *state* yang bertemu pada suatu *state* yang sama. Jalur *state* dengan total jarak hamming yang lebih rendah akan dilewatkan ke proses selanjutnya, sedangkan *state* dengan total jarak hamming yang lebih tinggi akan tereliminasi.
Sebuah *butterfly* secara lebih detail akan memiliki bentuk sebagai berikut :



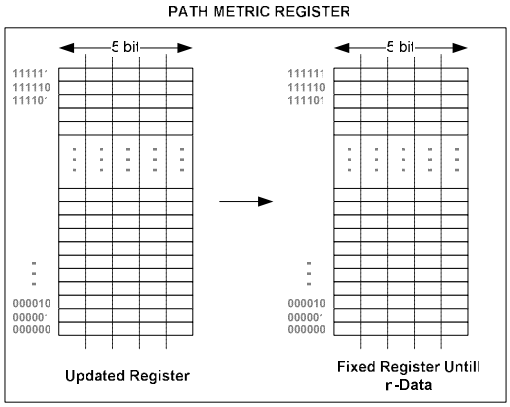
Gambar 7 Arsitektur *Butterfly*

Path_In_Up dan *Path_In_Down* merepresentasikan *state* awal sebagai *state* masukan, sedangkan Path_Out_Up dan Path_Out_Down menunjukkan arah perpindahan *state* masukan. Sebuah *butterfly* terbangun atas dua buah wing *butterfly*. Sebuah wing *butterfly* dibangun dari *branch metric*, adder, dan compare. *Branch metric* melakukan proses penghitungan jarak hamming yang merupakan perbedaan antara input pendekode dengan keluaran pengkode pada suatu transisi *state*.
Hasil proses yang dilakukan oleh *butterfly* ini selanjutnya akan disimpan dalam memori. Memori akan menyimpan *state* akhir dari sejumlah data yang diproses dan nilai *path metric* terkecil dari jalur survive yang dihasilkan.

D. Memori *State Metric*

Jenis memori register ini berfungsi untuk menyimpan nilai *state metric* (akumulasi dari *branch metric – branch metric*) untuk setiap *state* yang terdapat pada diagram trellis. Jika terjadi pertemuan 2 buah *state* yang berbeda pada satu *state* tujuan, maka akan dilakukan pembandingan untuk selanjutnya dipilih salah satu jalur dengan

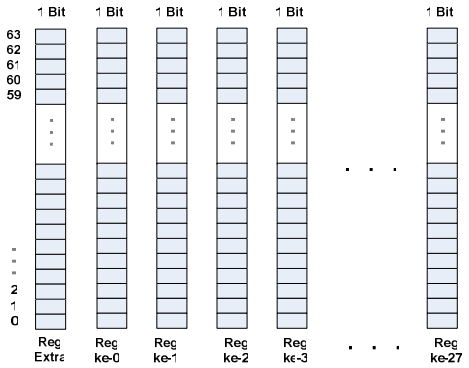
nilai *path metric* yang lebih kecil. Struktur memori *state metric* ini terdiri atas sepasang register dengan ukuran masing-masing 64 x 5 bit. Angka 64 menunjukkan jumlah *state* keseluruhan yang terdapat pada pengkode konvolusi dengan *constrain length* = 7. Sedangkan ukuran 5 bit dipilih dengan anggapan bahwa nilai tertinggi yang diasumsikan dapat muncul pada proses penghitungan *state metric* adalah 31 yang berasal dari 2^5.
Bentuk arsitektur memori *state metric* yang dirancang adalah sebagai berikut :



Gambar 8 Desain Memori *State Metric*

E. Memori Jalur *Survive*

Memori jalur *survive* ini berukuran 28 x 64 bit. Jika register terisi nilai '0', maka path yang survive berasal dari jalur *state* yang lebih tinggi. Sedangkan register akan terisi nilai '1' jika path yang survive berasal dari jalur *state* yang lebih rendah. Pola bit yang terisi pada register selanjutnya akan dimanfaatkan pada proses *traceback*.



Gambar 9 Desain *Path storage* Register dengan 1 *Butterfly*

F. Blok Generator Keluaran *Decoder*

Konsep *traceback* yang dikembangkan dalam penelitian ini adalah sebagai berikut :

1. Register *path metric* akan menghitung *path metric* di setiap *state* hingga data ke-28.
2. Selanjutnya nilai *path metric* yang terdapat pada setiap *state* akan dibandingkan dan diambil nilai minimal.
3. Nilai minimal ini selanjutnya akan dikirimkan ke *blok kontrol* beserta informasi pada *state* berapa nilai minimal ini dicapai.
4. Setelah seluruh isi register *path storage* terisi, pada register ke-27, bedasarkan informasi yang dikirim dari register *path metric*, *blok kontrol* akan mengirimkan informasi *state* yang memiliki nilai *path metric* minimal.
5. *State* ini pada register ke-27 akan menjadi titik awal *traceback*.
6. Jika bernilai *path storage* bernilai '0', maka *state* sebelumnya berasal dari jalur *state* yang lebih tinggi, sedangkan jika bernilai '1', maka *state* sebelumnya berasal dari jalur *state* yang lebih rendah.
7. Setelah identifikasi *state* asal diperoleh, nilai kode informasi adalah nilai LSB dari konversi nilai *state* pada basis binernya.

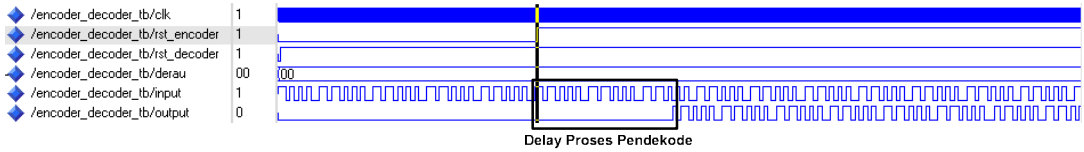
G. Blok Antarmuka Keluaran
Setelah informasi hasil pendekode di peroleh melalui proses taceback, maka selanjutnya data akan dikeluarkan pada interface keluaran. Fungsi untuk mengeluarkan data secara berurutan dari hasil proses *traceback* akan dilakukan oleh blok parallel to serial. Data hasil pendekodean yang diperoleh harus diurutkan secara serial mulai dari hasil pendekodean data pertama pada register ke-27 hingga data terakhir pada register ke-0.

3.3 *Synthesis*
Proses ini dilakukan untuk mendapatkan bentuk realisasi rangkaian yang telah dideskripsikan dengan VHDL. *Synthesis* dilakukan dengan menggunakan design analyzer dengan class library *synthesis* dari synopsys *synthesis tools*.

3.4 *Layout*
Langkah dalam desain setelah sintesis diselesaikan adalah menentukan bentuk implementasi rangkaian hasil sintesis dalam teknologi yang ditargetkan. Tahapan ini biasa dikenal sebagai desain fisik atau *layout* sintesis. Dalam *layout*, terdapat dua bagian utama yang dilakukan yaitu *placement* dan *routing*.
Placement merupakan tahap penentuan letak pada target devais setiap fungsi logika akan direalisasikan pada rangkaian yang dioptimasi. Tugas *placement* ini sangat bergantung pada teknologi implementasi yang digunakan.
Setelah tahapan *placement* selesai, akan dilanjutkan dengan realisasi interkoneksi yang dibutuhkan pada chip. Tahapan ini yang dikenal sebagai *routing*.

4. HASIL DAN PEMBAHASAN
Pengujian yang dilakukan berhubungan dengan hasil simulasi yang dilakukan dengan bahasa VHDL pada perangkat lunak modelsim. Pengujian pada hasil simulasi dilakukan dengan memberikan sinyal test vector pada arsitektur pengkode dan pendekode yang telah didesain. Pengujian dengan memberikan sinyal test vector ini dikenal sebagai testbench. Selain pengujian fungsional sistem dengan menggunakan modelsim, dilakukan juga proses sintesis rangkaian berdasarkan program VHDL yang telah disimulasikan.

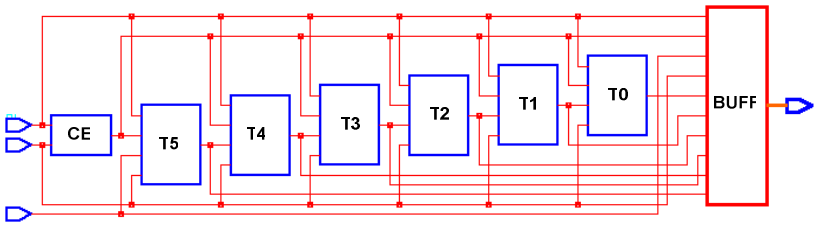
4.1 Pengujian Keseluruhan Sistem Pengkode dan Pendekode
Pada top level arsitektur pengkode dan pendekode, hasil running program testbench yang diperoleh adalah sebagai berikut :



Gambar 10 Delay Proses Pendekode Dengan 1 *Butterfly*

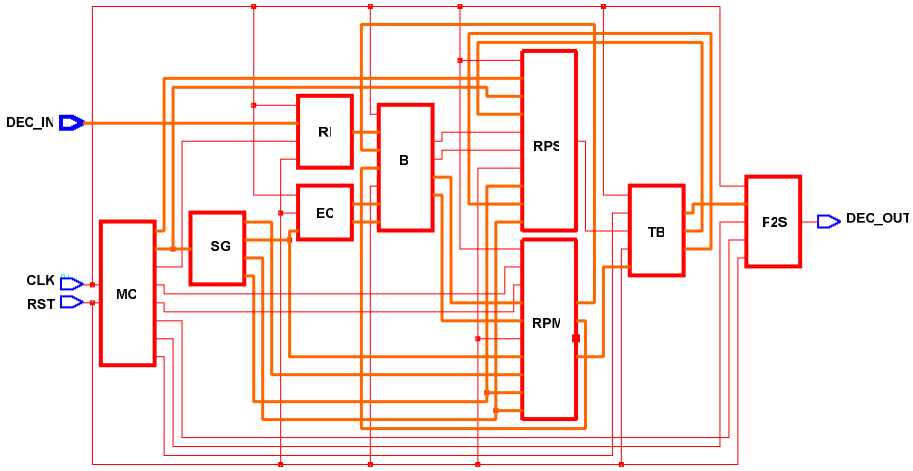
Berdasarkan sinyal keluaran testbench pada gambar 10 di atas, diperoleh delay proses pendekode sebesar 20.040 ns yang ekivalen dengan 1.020 *clock*.

4.2 Proses Sintesis Pengkode Konvolusi dan Pendekode Viterbi
Sintesis dilakukan secara berulang-ulang dengan mencoba nilai perioda *clock* yang berbeda-beda pada setiap kali sintesis dilakukan. Tujuan dari perlakuan ini adalah untuk mendapatkan nilai perioda *clock* terkecil yang masih dapat digunakan pada rangkaian yang telah didesain. Parameter acuan nilai *clock* terkecil yang masih dapat digunakan adalah pada nilai *slack*. Nilai *slack* menunjukkan sisa waktu dari satu perioda *clock* pada saat rangkaian bekerja pada suatu nilai *clock* tertentu. Periode *clock* terkecil yang diujikan pada percobaan ini sebesar 3 ns (333 MHz). Pemilihan nilai perioda *clock* sebesar 3 ns ini berdasarkan hasil sintesis bagian timing dimana pada nilai 3 ns masih tersisa waktu proses (*Slack*) untuk arsitektur pendekode dengan 1 *butterfly* sebesar 0,01 ns. Sedangkan untuk perioda < 3ns (kurang dari 3 ns), nilai *slack* akan berharga negatif. Nilai negatif ini menunjukkan bahwa waktu proses yang diperlukan tidak dapat dipenuhi oleh periode pulsa *clock* yang tersedia.
Pada proses sintesis rangkaian, bagian blok encoder menghasilkan total cell area sebesar 1.450 μm^2 atau ekivalen dengan 0,00145 mm^2. Sedangkan hasil sintesis untuk blok decoder viterbi dengan arsitektur menggunakan 1 *butterfly* membutuhkan total cell area sebesar 295.431 μm^2 atau ekivalen dengan 0,29543 mm^2.
Hasil sintesis rangkaian pengkode dan pendekode yang dihasilkan adalah sebagai berikut :



Gambar 11 Hasil Sintesis Rangkaian Pengkode Konvolusi

Keterangan :
CE = *Clock* Enable
T5 – T0 = Blok Tunda ke-5 sampai dengan Tunda ke-0
BUFF = Buffer



Gambar 12 Hasil Sintesis Rangkaian Pendekode dengan 1 *Butterfly*

Keterangan :
RI = Register Input
SG = State Generator
MC = Main Control
B = *Butterfly*
RPM = Register Path Metric
RPS = Register Path Storage
TB = Trace Back
P2S = Paralel to Serial

Secara lengkap hasil sintesis yang dilakukan untuk berbagai perioda *clock* ditunjukkan pada table berikut ini :

Tabel 1 Hasil Sintesis dengan Menggunakan 1 *Butterfly*

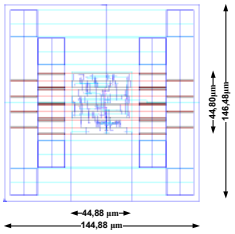
PERIODA	***ENCODER***		***DECODER***	
***CLOCK* (ns)**	**AREA (μm^2)**	***SLACK* (ns)**	**AREA (μm^2)**	***SLACK* (ns)**
3	1,450	2.22	295,866	0.01
4	1,450	3.22	295,364	1.19
5	1,450	4.22	295,231	2.05
6	1,450	5.22	295,547	3.01
7	1,450	6.22	295,617	3.96
8	1,450	7.22	295,845	4.98
9	1,450	8.22	295,121	6.09
10	1,450	9.22	295,647	7.14
12	1,450	11.22	295,514	9.02
15	1,450	14.22	295,627	11.92
18	1,450	17.22	295,317	15.01
20	1,450	19.22	294,479	16.98

Berdasarkan tabel di atas, *slack* pada bagian decoder selalu lebih kecil daripada *slack* pada bagian encoder untuk setiap perioda *clock* yang diujikan. Hal ini menunjukkan kompleksitas rangkaian pada bagian decoder lebih tinggi daripada bagian encoder. Sehingga waktu pemrosesan hingga diperoleh keluaran akhir yang terjadi pada bagian decoder akan lebih besar daripada waktu yang dibutuhkan oleh encoder. Dengan kebutuhan waktu pemrosesan lebih besar, maka *slack* menjadi lebih kecil.

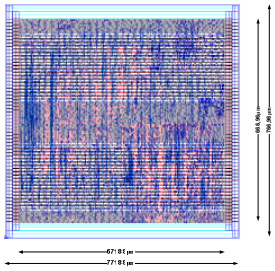
4.3 Proses *Layout*

Layout dilakukan dengan menggunakan library standard teknologi *CMOS* 0,18 mikron dengan perangkat lunak dari synopsys. Untuk *layout*, dilakukan secara terpisah antara bagian *encoder* dan *decoder*.

Berdasarkan serangkaian proses *layout* yang telah dilakukan, tampilan *layout* chip yang diperoleh untuk pengkode dan pendekode adalah sebagai berikut :



Gambar 13 Tampilan *Layout* Chip Pengkode Konvolusi



Gambar 14 Tampilan *Layout* Chip Pendekode Konvolusi dengan 1 *Butterfly*

Parameter hasil layout yang diperoleh ditunjukkan pada tabel berikut :

Tabel 2 Hasil *Layout* Pada Standard Teknologi CMOS 0,18 *Mikron*

NO.	PARAMETER	ENCODER	DECODER
1.	Frekuensi Kerja (MHz)	100	100
2.	Kecepatan Data (Mbps)	3,125	3,125
3.	Ukuran Core (mm^2)	0,00201	0,44812
4.	Ukuran Chip (mm^2)	0,21222	0,59200
5.	Rasio cell/core (%)	72,1324	65,919
6.	Rasio cell/chip (%)	6,83399	49,8976

Berdasarkan tabel parameter layout di atas, jika dibandingkan antara frekuensi kerja hasil sintesis sebesar 333 MHz, maka terjadi penurunan sebesar 233 MHz. Hasil layout yang terdiri dari proses floorplanning dan routing pada proses implementasi dalam pendekatan IC (Integrated Circuit) design menghasilkan frekensi kerja sebesar 100 MHz. Dengan frekuensi kerja sebesar 100 MHz, maka kecepatan data yang dapat didukung oleh rangkaian pendekode viterbi menggunakan 1 butterfly ini sebesar 3,125 Mbps. Hasil ini diperoleh dari persamaan sebagai berikut :

$$\text{Kecepatan Data} = \frac{FrekuensiKerja}{\sum ButterflyNormal} \quad (4)$$

Dengan frekuensi keja = 100 MHz dan ΣbutterflyNormal untuk nilai constrain length pada penelitian ini sebanyak 32 butterfly, maka diperoleh hasil kecepatan data yang dapat dilewatkan pada sistem pendekode viterbi ini adalah 3,125 Mbps.

5. KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan dalam tesis ini dapat disimpulkan beberapa hal sebagai berikut :

1. Perancangan pengkode konvolusi dan pendekode viterbi telah dapat dilakukan pada tahap simulasi fungsional sistem, sintesis rangkaian hasil disain. Perancangan bagian pendekode viterbi dilakukan dengan arsitektur menggunakan 1 *butterfly*.
2. Hasil sintesis yang dilakukan menggunakan design analyzer perangkat lunak synopsys menghasilkan area sintesis untuk pengkode konvolusi sebesar 1.450 μm^2 atau ekivalen dengan 0,00145 mm^2. Sedangkan hasil sintesis bagian pendekode viterbi menggunakan 1 *butterfly* menghasilkan area rata-rata sebesar 29.5431 μm^2 atau ekivalen dengan 0,29543 mm^2. Perioda *clock* terkecil yang masih dapat diterapkan pada sintesis pengkode konvolusi dan pendekode viterbi menggunakan 1 *butterfly* adalah 3 ns.
3. Proses layout pada standard teknologi CMOS 0,18 mikron telah berhasil dilakukan dengan frekuensi kerja pendekode viterbi hasil layout sebesar 100 MHz. Dan kecepatan data yang dapat diaplikasikan pada pendekode viterbi menggunakan 1 butterfly sebesar 3,125 Mbps.

6. DAFTAR PUSTAKA

Wolf, Wayne, 1998, *Modern VLSI Design*, Prentice Hall PTR.

Proakis, John G. , 1995, *Digital Communications*, McGraw-Hill *International Editions*, *Third Edition*.

Brown, Stephen, Zvonko Vranesic, 200, *Fundmentals of Digital Logic*, McGraw-Hill *International Edition*3.

Chang, K. C. 1999, *Digital Systems Design with VHDL and Synthesis*, IEEE *Computer Society*,.

Angarita, F., A. Perez-Pascual, T. Sansaloni, J. Valls, *Efficient Mapping on FPGA of a Viterbi Decoder for Wireless LANs*, Departamento de Ingenieria de Valencia, Spain.

4121 *Communications* Ltd, 2001, *Convolutional Encoder and Viterbi Decoder*, *Revision* 3.1.

Morelos, Robert H., Zaragoza, 2002, *The Art of Error Correcting Coding*, John Wiley & Sons.

Lin, Shu, Daniel J. Costello, Jr., *Error Control Coding*, 1983, Prentice-Hall Series in Computer Applications in Electrical Engineering.