PERANCANGAN VLSI ARSITEKTUR YANG OPTIMUM MENGGUNAKAN *LOW LEVEL ABSTRACTION* MODEL SYSTEMC DAN METODE VERIFIKASI BERBASIS *ASSERTION* PADA MODUL *TURBO CODE* UNTUK LTE

LAPORAN TUGAS AKHIR

Oleh :
ARDIMAS ANDI PURWITA
13207044

PROGRAM STUDI TEKNIK ELEKTRO
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2011

PERANCANGAN VLSI ARSITEKTUR YANG OPTIMUM MENGGUNAKAN *LOW LEVEL ABSTRACTION* MODEL SYSTEMC DAN METODE VERIFIKASI BERBASIS *ASSERTION* PADA MODUL *TURBO CODE* UNTUK LTE

LAPORAN TUGAS AKHIR

Diajukan sebagai salah satu syarat untuk
memperoleh gelar Sarjana Teknik
Program Studi Teknik Elektro
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Oleh :
ARDIMAS ANDI PURWITA
13207044

PROGRAM STUDI TEKNIK ELEKTRO
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2011

PERANCANGAN VLSI ARSITEKTUR YANG OPTIMUM MENGGUNAKAN *LOW LEVEL ABSTRACTION* MODEL SYSTEMC DAN METODE VERIFIKASI BERBASIS *ASSERTION* PADA MODUL *TURBO CODE* UNTUK LTE

Oleh :
ARDIMAS ANDI PURWITA
13207044

LAPORAN TUGAS AKHIR

telah diterima dan disahkan
untuk memperoleh gelar Sarjana Teknik
Program Studi Teknik Elektro
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Bandung, 02 Agustus 2011

Pembimbing

Trio Adiono,Ph.D
NIP 132163103

ABSTRAK

Laporan tugas akhir ini berisi tentang optimasi dan implementasi model fungsional dengan *low level abstraction* yaitu *Register Transfer Level* (RTL) menggunakan systemC dan verifikasi berbasis *assertion* (ABV) pada modul kode turbo (*turbo code*) untuk *Long Term Evolution* (LTE). Fokus utama dari tugas akhir ini adalah optimasi kode turbo untuk LTE dan metode perancangan sistemnya, yaitu pemodelan fungsional RTL menggunakan systemC, perancangan *hardware* (HW) menggunakan *Hardware Description Language* (HDL), dan metode verifikasi HW menggunakan ABV.

Pada tahap optimasi perancangan kode turbo, masalah utama yang dimiliki adalah waktu latensi pemrosesan yang panjang, baik untuk enkoder maupun dekoder, serta tingginya kompleksitas perhitungan dan kebutuhan memori pada komponen dekodernya. Panjangnya waktu latensi kode turbo ini karena sifatnya yang iteratif dalam proses mendeteksi kesalahan (*error*). Pada tugas akhir ini dirancang kode turbo dengan 4 kali iterasi. Teknik paralelisasi 8-level digunakan untuk memperpendek waktu latensi dan dilakukan juga optimasi dari hasil paralelisasi tersebut pada beberapa blok, sehingga didapat *trade off* yang optimal. Pada tugas akhir ini digunakan pula kombinasi algoritma Max-log-MAP dan *Sliding Window Algorithm* (SWA) untuk mengurangi tingginya kompleksitas perhitungan sekaligus memperpendek waktu latensi dan penggunaan memori pada komponen dekoder. Namun terdapat satu hal yang penting yang perlu diperhatikan dalam proses optimasi kode turbo ini, yaitu performa *Bit Error Rate* (BER). Performa BER kode turbo yang telah dioptimasi dibandingkan dengan kode turbo acuan agar didapatkan performa BER yang sama baiknya atau lebih baik.

Tugas akhir ini menggunakan metode perancangan yang mendukung sehingga HW dan *software* (SW) dapat disimulasikan bersamaan dalam satu *environment* yang menggabungkan pemodelan fungsional menggunakan systemC, implementasi HW menggunakan HDL (verilog), dan metode verifikasi HW-nya yaitu ABV menggunakan System Verilog Assertion (SVA). Pemodelan fungsional dengan menggunakan systemC mendukung beberapa level abstraksi yang dinyatakan dalam tingkat keakuratan port dan waktu. Setelah memodelkan sistem secara fungsional menggunakan systemC, dilakukakn proses translasi model ke HW menggunakan HDL. Proses translasi HW dilakukakn karena belum ada *tool* yang dapat menyintesis dari model systemC dengan optimum. Agar proses translasi dari model fungsional ke HW tidak memiliki gap yang

jauh, maka pada tugas akhir ini, dirancang model fungsional yang memiliki tingkat akurasi port dan waktu yang mendekati implementasi HW, yaitu model RTL.

Tugas akhir ini juga mencakup proses sintesis pada FPGA ALTERA Cyclone II EP2C3-5F672C6 untuk memastikan apakah kode turbo yang dirancang memenuhi frekuensi kerja LTE, yaitu mencapai hingga 100 Mhz pada *downlink*. Pada tugas akhir ini berhasil diimplementasikan model fungsional RTL, implementasi HW-nya, dan ABV. Dilihat dari hasil sintesisnya, dapat disimpulkan bawa kode turbo yang telah dirancang dapat bekerja pada frekuensi maksimum hingga 115.86 Mhz. Sedangkan, dari hasil implementasi algoritma Max-log-MAP, SWA, dan 8-level paralelisasi yang telah dioptimasi pada kode turbo ini , didapatkan ukuran memori hanya sebesar 16 KB dan waktu latensi berhasil diperpendek hingga maksimum sekitar 8300 *clock*. Performa BER-nya pun memiliki hasil yang lebih baik pula.

Kata Kunci: optimasi, paralel, LTE, RTL, verilog, systemC, SVA

ABSTRACT

This final year project is about the optimization and implementation of low level abstraction (Register Transfer Level (RTL)) functional model using systemC and assertion based verification (ABV) of turbo code on Long Term Evolution (LTE). The main focuses of this final year project are the optimization turbo code on LTE and system design method of turbo code, that is functional model using systemC, hardware (HW) design using Hardware Description Language (HDL), and HW verification using ABV.

In the optimization design of turbo code, the main problems are long latency of tubo code (turbo encoder and decoder), high complexity of computation and excessive memory usage on its decoder component. This long latency is caused by iterative process for correcting the errors. In this final year project, tubo code has four times iteration. By using 8-level paralellization technique, the long latency of turbo code can be shortened and the more optimization is applied for some blocks. Therefore, the optimum trade-off can be obtained. In this final year project, the combination of Max-log-MAP algorithm and Sliding Window Algorithm (SWA) is used to reduce the high complexity of computation its decoder component and shorten high latency and memory usage at once. However, The Bit Error Rate (BER) performance needs to be considered also. This performance is compared with the reference turbo code in order to obtain the same or better performance.

For system design level, the hardware-software (SW) co-simulation is built in this final year project which can be executed in one environment thas combines functional model using systemC, implementation HW using verilog, and HW verification method which is assertion based verification using System Verilog Assertion (SVA). After, functional model using systemC has been applied, that model using systemC translate into HW using HDL. This translation process is done because still there are no tools that support optimized synthesizable systemC. The functional model using systemC supports many abstraction levels which can be usually measured by accuration of port and time. In order to simplify translation process from functional model to HW which has far gap, in this final year project, functional model which has the most accurate port and time which is RTL.

The HW that has been implemented by HDL is synthesized on FPGA ALTERA Cyclo-

ne II EP2C35F672C6 in orde to make sure that turbo code supports frequency up to 100 Mhz (the minimum frequency of LTE on downlink). In this final year project, the RTL functuinal model, its HW implementation, and ABV is successfully designed. By analyzing the synthesis report of HW implementation of optimized turbo code, the maximum frequency that can be applied is 115.85 MHz. Whereas, from HW implementation result, memory of decode component has 16 KB and the shorter latency (maximal approximately 8300 clocks). The BER performance has better result also.

Keywords: optimize, parallel, LTE, RTL, verilog, systemC, SVA

KATA PENGANTAR

Alhamdulillah. Segala puji dan syukur penulis panjatkan kepada Allah SWT atas limpahan rahmat dan nikmat-Nya sehingga penulis dapat menyelesaikan laporan Tugas Akhir dengan judul "**PERANCANGAN VLSI ARSITEKTUR YANG OPTIMUM MENGGUNAKAN *LOW LEVEL ABSTRACTION* MODEL SYSTEMC DAN METODE VERIFIKASI BERBASIS *ASSERTION* PADA MODUL *TURBO CODE* UNTUK LTE**". Tugas akhir ini merupakan syarat kelulusan untuk menjadi sarjana dan merupakan rangkaian terakhir dari perjalan menempuh ilmu di ITB. Melalui tugas akhir ini, diharapkan menjadi langkah nyata awal bagi penulis unttuk menggapai cita-citanya sehingga penulis dapat menjadi bagian penting dalam kemajuan teknologi di bidang *IC design*. Dalam pengerjaan tugas akhir ini, penulis tidak dapat terlepas dari bantuan banyak pihak. Oleh karena itu, pada kesempatan ini penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

- KELUARGA tercinta yang setia mendukung dan menjadi motivasi terbesar bagi penulis;
- TRIO ADIONO, PH.D sebagai pembimbing dan *role model* yang menginspirasi penulis untuk benar-benar berusaha di bidang *IC design*;
- PAK YUDI, PAK BUDI, DAN BU ELVA telah menguji selama hampir satu setengah jam tetapi tetap menyenangkan. Semoga masukan2 yang diberikan dapat membawa manfaat bagi penulis ;
- AULIA AYUNINGTYAS, yang selalu menemani, mendukung, dan mengingatkan penulis selama pengerjaan tugas akhir ini ;
- RELLA MARETA yang pernah mengingatkan tujuan utama hidup ini sebenarnya apa, sehingga di detik terakhir dalam proses pembuatan laporan tugas akhir ini penulis merasa tidak tegang dan bersikap lebih santai ;
- YANWAR ARDITIYAS DAN PAULUS CUANG (TIM PUTRA GANESHA), sebagai teman kelompok dalam LSI design Okinawa yang membantu menunjukkan minat utama penulis di bidang *IC design*;

- YAYAN, AAN, KEN, DAN ARNAUD, yang telah membantu secara teknis dalam proses memahami spesifikasi pada tugas akhir ini;
- TEMAN-TEMAN LAB. IC DESIGN, AVRG, DAN KPRG yang telah memberikan suasana tugas akhir sehingga penulis terus termotivasi dalam mengerjakan tugas akhir ini;
- Semua teman Workshop HME ITB dan HME ITB yang telah memberi pengalaman baik *hard skill* maupun *soft skill*;
- Semua teman STEI angkatan 2007 yang tidak dapat disebutkan satu per satu;
- Semua teman yang dikenal tapi belum disebutkan namanya.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Keunggulan kemampuan *Forward Error Correction* (FEC) dari kode turbo menjadikannya bagian dari mayoritas standar komunikasi yang digunakan saat ini, termasuk *High-Speed Downlink Packet Access* (HSDPA) dan *Long-Term Evolution* (LTE) yang menggunakan standar komunikasi *3rd Generation Partnership Project* (3GPP) *Universal Mobile Telecommunications System* (UMTS) [**?**]. Claude Berrou dan Alain Glavieux menjelaskan keunggulan utama dari kode turbo adalah kemampuan kode turbo yang mendekati prediksi teori batas (*theoretical limit predicted*) oleh Shannon [**?**]. Teori tersebut mendefinisikan kecepatan data maksimum yang dapat ditransfer melalui sebuah kanal dengan kesalahan yang minimum [**?**].

Semenjak Shannon mempublikasikan teorinya tersebut pada tahun 1948, banyak penelitian tentang FEC yang telah dilakukan untuk mendekati hasil prediksi teori batas Shannon, misalnya kode Hamming [**?**], konvolusional kode FEC [**?**], algoritma Viterbi [**?**], kode BCH [**?**], dll. Namun, diantara metode-metode tersebut, kode turbo yang ditemukan oleh Berrou, Glavieux, dan Thitimajshima adalah metode yang paling mendekati teori Shannon [**?**][**?**]. Kode ini terdiri dari dua paralel komponen enkoder atau dekoder beserta blok *interleaver* dan *deinterleaver*.

Dikarenakan alasan tersebut, sejak penemuan kode turbo, banyak penelitian lebih lanjut yang telah dilakukan hingga kode turbo mencapai keadaan stabil (*state of maturity*) [**?**]. Oleh karenanya kode turbo ini banyak dipakai pada standar komunikasi yang ada pada saat ini. Salah satu contohnya adalah LTE yang dikembangkan oleh 3GPP. LTE ini menjadi solusi dari tuntutan pengguna komunikasi dijital akibat kekurangan teknologi pendahulu LTE, yaitu *Global System for Mobile Communications* (GSM)/ *Enhanced Data rates for GSM Evolution* (EDGE) dan *Universal Mobile Telecommunications System* (UMTS)/ *High Speed Packet Access* (HSPA) [**?**]. LTE diharapkan dapat memiliki kecepatan yang lebih tinggi dan mengurangi latensi yang dirasakan oleh pengguna teknologi sebelumnya dengan kemampuan pengiriman data mencapai kecepatan 100 Mbit/s (*downlink*) atau 50 Mbit/s (*uplink*), serta meningkatkan sisi mo-

bilitasnya. LTE juga dijadwalkan untuk menyediakan dukungannya kepada teknologi berbasiskan *Internet Protocol* (IP).

Namun, seiring dengan keunggulan dari kode turbo, bukan berarti kode turbo ini tidak memiliki kekurangan. Jika dibandingkan dengan turbo enkoder, turbo dekoder memiliki tingkat kompleksitas yang lebih tinggi. Sehingga, banyak penelitian lebih berfokus pada implementasi turbo dekoder. Dua kekurangan utama yang dimiliki turbo dekoder adalah tingginya tingkat kompleksitas perhitungan dan konsumsi energi yang tinggi termasuk jika dibandingkan kode konvolusional [**?**]. Oleh karena itu, muncul banyak algoritma sebagai jalan tengah antara keunggulan performa kode turbo dengan tingkat kompleksitas perhitungan yang dimilikinya. Pada dasarnya, algoritma-algoritma yang ada merupakan pengembangan dari *Soft-Output-Viterbi Algorithm* (SOVA) dan *Maximum A posteriori Probability* (MAP) *algorithm*. Diantara keduanya, algoritma MAP memiliki performa yang lebih mendekati hasil prediksi Shannon [**?**] [**?**].

Algoritma MAP ini memiliki beberapa kekurangan pada tingginya kompleksitas perhitungan seperti yang telah dijelaskan sebelumnya jika diimplementasikan menjadi sebuah HW. Tingginya kompleksitas perhitungan terdapat pada operator logaritma natural, selain itu penggunaan memorinya yang boros. Sehingga muncullah algoritma-algoritma yang melakukan pendekatan pada penggunaan operator ini, yaitu Algoritma log-MAP dan max-log-MAP [**?**]. Sedangkan, untuk permasalahan penggunaan memori, SWA [**?**] menjadi solusi untuk mengatasinya. Pada tugas akhir ini, kombinasi algoritma max-log MAP dan SWA digunakan sebagai landasan optimasi yang dilakukan dalam perancangan turbo dekoder. Selain itu, untuk turbo enkoder juga dilakukan optimasi sehingga didapatkan kecepatan pengolahan data yang lebih tinggi dan latensi waktu yang lebih pendek.

Pada perancangan HW, penggunaan HDL (verilog, VHDL, AHDL, dll) semakin banyak digunakan karena semakin banyak *tools*(*tools* simulasi dan sintesis) pendukung. Proses perancangan HW pada awalnya terpisah dengan perancangan SW dan keduanya akan digabungkan jika sudah mencapai tahap akhir. Dengan metode ini, proses perancangan secara keseluruhan menjadi lambat, sehingga dibutuhkan metode yang mendukung kedua perancangan ini dilakukan secara bersamaan. Istilah yang sering digunakan untuk pendekatan perancangan secara bersamaan dan multi disiplin ini adalah *Electronic System-Level* (ESL). Contoh dari *environtment* yang mendukung skenario perancangan seperti demikian adalah systemC. SystemC adalah sebuah *class* dan *template library* untuk bahasa pemograman C++ sehingga dapat menyimulasikan HW. Namun, sampai saat ini masih dilakukan pengembangan untuk menyintesisnya agar

mendapatkan implementasi HW yang efisien [**?**].

SystemC mendukung berbagai level model abstraksi. Dari *un-timed* model hingga port dan waktu yang akurat (RTL). Pada tugas akhir kali ini, kode turbo akan dimodelkan pada level abstraksi yang paling rendah yaitu RTL yang memiliki tingkat akurasi port dan waktu paling mendekati HW. Fitur tersebut menjadi solusi terhadap lebarnya gap antara bahasa pemograman yang digunakan untuk model fungsional dengan HW [**?**]. Dari model yang telah dibuat tersebut, kemudian dilakukan proses translasi ke HW menggunakan HDL (verilog). Proses ini dilakukan karena masih belum ada *tool* yang dapat langsung menyintesis model systemC yang optimum [**?**].

Sebelum mencapai tahap implementasi, dari rancangan yang telah dibuat hingga menjadi sebuah chip, maka perlu dilakukan beberapa tahap verifikasi. Pada kenyataanya setiap tahapan pembuatan sebuah chip memiliki teknik verifikasinya masing-masing, dari verifikasi model hingga verifikasi manufaktur [**?**]. Verifikasi ini menjadi isu penting dalam pembuatan sebuah chip karena menghabiskan 70% konsumsi waktu, biaya, dan tenaga [**?**]. Dengan kompleksitas yang tinggi, ukuran yang kecil, dan waktu produksi yang pendek, dituntut adanya metode verifikasi yang paling efisien dan efektif agar dapat menghasilkan chip berkualitas tinggi dengan biaya yang sedikit mungkin.

Metode verifikasi konvensional membutuhkan waktu yang lama karena baik stimulus input maupun hasil referensi yang harus dibandingkan berasal dari file *dump* model fungsional yang telah dirancang sebelumnya. Selain itu, diperlukan keuletan dalam melihat jendela yang disediakan oleh *tools* untuk memeriksa jika terjadi kesalahan dalam merancang. Metode ini semakin ditinggalkan karena memiliki pendekatan verifikasi *black box* yang memiliki tingkat observabilitas yang rendah. Oleh karena itulah, metode ABV [**?**] [**?**] diperkenalkan untuk mengatasi masalah-masalah tersebut. Dalam tugas akhir ini, akan diimplementasikan ABV sebagai metode verifikasi perancangan HW.

1.2. Rumusan Masalah

Berdasarkan latar belakang tersebut, ada beberapa aspek yang perlu diperhatikan dalam pengerjaan tugas akhir ini, yaitu waktu pemrosesan yang dilakukan kode turbo, kompleksitas perhitungan serta ukuran memori yang dibutuhkan, metode perancangan sistem yang optimal (simulasi bersama HW dan SW), dan metode verifikasi yang digunakan.

Kode turbo memiliki beberapa masalah jika diimplementasikan menjadi HW. Masalah yang pertama adalah tingginya latensi waktu yang dibutuhkan. Masalah yang kedua adalah kompleksitas perhitungan untuk mengimplementasi turbo dekoder. Masalah yang ketiga adalah besarnya ukuran memori yang dibutuhkan pada komponen dekoder.

Untuk merancang sebuah sistem yang lengkap (HW dan SW) dibutuhkan waktu yang lama jika menggunakan metode konvensional (perancangan HW dan SW terpisah dan digabungkan di tahap akhir). Implementasi simulasi bersama antara HW dan SW menjadi isu pada tugas akhir ini. Kemudian masalah terakhir yang dibahas pada tugas akhir ini adalah implementasi metode verifikasi HW yang digunakan yaitu ABV, sehingga didapatkan parameter yang lebih jelas bahwa rancangan HW yang telah dibuat benar dibandingkan dengan metode verifikasi konvensional.

Dari latar belakang dan masalah yang telah dijelaskan, rumusan masalah dari tugas akhir ini adalah sebagai berikut:

1. Bagaimana merancang arsitektur dan mengimplementasikan algoritma pada kode turbo sehingga didapatkan waktu pemrosesan (latensi) yang lebih pendek, kompleksitas perhitungan yang lebih rendah, kebutuhan memori yang lebih sedikit, dan memiliki performa BER sama baiknya atau lebih baik dibandingkan kode turbo referensi?
2. Bagaimana mengimplementasikan *environtment* yang mendukung simulasi bersama HW menggunakan HDL dan SW (model fungsional RTL menggunakan systemC)?
3. Bagaimana menerapkan metode verifikasi HW menggunakan ABV sehingga didapatkan parameter keberhasilan yang lebih jelas dibandingkan metode verifikasi konvensional?

1.3. Tujuan

Tujuan pelaksanaan tugas akhir ini adalah:

1. Rancangan turbo dekoder dapat bekerja dengan total *clock* yang dibutuhkan kurang dari 10000 *clock* dan frekuensi minimum 100 MHz, sedangkan untuk kode enkoder memiliki *clock* kurang dari 10000 *clock* dan frekuensi minimum 100 Mhz.

2. Implementasi komponen dekoder memiliki perhitungan yang lebih sederhana dibanding operator logaritma natural.
3. Implementasi komponen dekoder memiliki memori tidak lebih dari 32 KB.
4. Implementasi *environtment* yang mendukung simulasi bersama HW menggunakan HDL (verilog) dan SW (model fungsional RTL menggunakan systemC).
5. Implementasi metode verifikasi HW yang digunakan yaitu ABV dengan semua propertinya berhasil dipenuhi (tidak ada yang *fail*).
6. HDL yang dibuat dapat disintesis pada FPGA Altera Cyclone II EP2C35F6-72C6.

1.4. Batasan Masalah

Tugas akhir ini dilakukan dengan batasan-batasan sebagai berikut:

1. LTE mengacu pada standar 3GPP TS 36.212 v9.2.0 [**?**].
2. Kode turbo fungsional yang menjadi referensi menggunakan program yang telah dibuat di Versatile Silicon Technologies (VST) dan tidak akan dibahas rinci pada tugas akhir ini.
3. Simulasi model fungsional (bukan RTL) dilakukan menggunakan MATLAB.
4. Simulasi dan verifikasi dilakukan menggunakan simulator ModelSim dari Mentor Graphics Corp.
5. RTL kode turbo menggunakan format data *fixed point*.
6. RTL yang didapat dari proses translasi dari systemC ke HDL tidak dinjau tingkat efisiensinya, hanya ditinjau fungsionalitasnya.
7. Target rancangan adalah board FPGA Altera Cyclone II EP2C35F672C6.
8. *Tool* sintesis yang digunakan adalah Quartus II 9.0.
9. RTL yang dibuat dapat disintesis pada FPGA Altera Cyclone II EP2C35F672C6.
10. Model saluran yang digunakan adalah modulasi *Bi-Phase Shift Keying* (BPSK) dengan model derau *Additive White Gaussian Noise* (AWGN).

1.5. Metodologi

Langkah pertama dalam metodologi pengerjaan tugas akhir ini adalah memahami spesifikasi dari desain. Hal pertama yang dilakukan dalam memahami spesifikasi desain ini adalah mempelajari teori kode turbo serta mempelajari teknik-teknik optimasi yang telah dilakukan sebelumnya. Kemudian dipelajari pula mengenai fitur-fitur yang terdapat di systemC dan teknik verifikasi ABV.

Langkah kedua adalah menerapkan optimasi yang telah dirancang pada model fungsional harus dioptimasi (didapat dari VST), lalu memverifikasinya. Tahap ini dilakukan dengan menggunakan bahasa MATLAB. Setelah itu dari model optimasi yang telah diimplementasikan dan diverifikasi, maka dilakukan proses translasi menjadi model fungsional RTL menggunakan systemC, lalu diverifikasi.

Kemudian, langkah selanjutnya sebelum mengubahnya ke HW menggunakan HDL (verilog), maka perlu didefinisikan dahulu tipe stimulus input acak, properti *assertion*, dan *functional coverage*-nya. Setelah itu, diimplementasikan bersamaan dengan proses translasi model systemC ke HW. Lalu, dilakukan verifikasi secara bersamaan antara model systemC dengan HW beserta ABV. Setelah berhasil, maka dilakukan proses sintesis sehingga memenuhi spesifikasi yang ditentukan di awal. Alur metodologi perancangan lengkapnya dapat dilihat pada Gambar. 1.1.

Gambar 1.1: Alur metodologi perancangan

1.6. Sistematika Penulisan

BAB I PENDAHULUAN
Bagian ini menjelaskan latar belakang, rumusan masalah, tujuan yang ingin dicapai, batasan masalah, metodologi perancangan, dan sistematika penulisan tugas akhir.

BAB II KODE TURBO
Pada bab ini kode turbo secara umum dan kode turbo sesuai standar teknis LTE yang dikeluarkan oleh 3GPP [**?**].

BAB III PEMODELAN FUNGSIONAL RTL MENGGUNAKAN SYSTEMC
Pada bab ini akan dibahas metode pemodelan fungsional RTL menggunakan systemC

dan tujuan penggunaannya pada tugas akhir ini. Selain itu, akan disampaikan pula beberapa peraturan umum proses translasi model fungsional RTL menggunakan systemC ke HW menggunakan HDL (verilog).

BAB IV VERIFIKASI BERBASIS *ASSERTION*

Pada bagian ini menjelaskan mengenai metode verifikasi HW yang digunakan, yaitu ABV beserta penjelasan *platform* dan keunggulan-keunggulannya.

BAB V OPTIMASI PERANCANGAN HARDWARE KODE TURBO

Bab ini akan memaparkan teknik atau algoritma dasar yang akan digunakan sebagai landasan dalam mengoptimasi kekurangan perancangan kode turbo menjadi HW dan optimasi yang dilakukan pada tugas akhir ini dengan diawali penjabaran kekurangan implementasi HW kode turbo ini.

BAB VI PERANCANGAN MODEL FUNGSIONAL KODE TURBO

Bagian ini menjelaskan implementasi model fungsional kode turbo dari optimasi yang telah dirancang pada bab 5. Perancangan model fungsional ini termasuk perancangan awal model menggunakan MATLAB dan perancangan model fungsional RTL menggunakan systemC.

BAB VII PERANCANGAN HDL DENGAN VERIFIKASI BERBASIS *ASSERTION*

Pada bab ini akan dilaporkan mengenai hasil translasi model fungsional RTL menggunakan systemC ke model RTL HW menggunakan HDL (verilog) dan implementasi ABV.

BAB VIII HASIL SIMULASI HARDWARE, SINTESIS, DAN ANALISIS KERJA

Pada bab ini akan ditampikan hasil simulasi HW lengkap beserta implementasi ABV dan hasil sintesis HWnya. Pada bab 8 akan ditampilkan pula hasil analisis dari kedua hasil ini.

BAB IX KESIMPULAN DAN SARAN

Bagian ini memuat kesimpulan dan saran dari pelaksanaan tugas akhir ini.

Gambar 2.1: Grafik perbandingan BER kode turbo dengan teori Shannon [**?**]

BAB 2

KODE TURBO

Pada bab ini akan dijabarkan mengenai kode turbo lebih rinci. Pembahasan bab ini dimulai dengan penjelasan mengenai kode turbo secara umum. Kemudian, pembahasan dilanjutkan mengenai kode turbo sesuai standar teknis yang dikeluarkan oleh 3GPP [**?**].

2.1. Pendahuluan Kode Turbo

Seperti yang telah disebutkan pada bab 1, teori batas Shannon membuktikan bahwa performa rata-rata kode yang dihasilkan secara acak dapat menghasilkan penurunan peluang kesalahan dekoder secara ekponensial dengan bertambahnya panjang blok enkoder yang digunakan. Namun demikian, Shannon tidak memberi petunjuk lebih bagaimana membuat kode tersebut.

Pada tahun-tahun berikutnya, setelah publikasi teori Shannon tersebut [**?**], banyak penelitian yang dilakukan untuk menciptakan kode yang memiliki performa mendekati teori batas Shannon. Sehingga struktur yang mendekati sifat acak (*randomlike properties*) Shannon pun mulai banyak ditemukan, seperti kode Hamming [**?**], konvolusional kode FEC [**?**], algoritma Viterbi [**?**], kode BCH [**?**], dll. Namun, diantara metode-metode tersebut, kode turbo yang ditemukan oleh Berrou, Glavieux, dan Thitimajshima adalah metode yang paling mendekati teori Shannon [**?**][**?**]. Kode ini berhasil memiliki struktur yang mendekati sifat acak Shannon dengan metode *decoding* iteratif yang efisien. Gambar 2.1 menunjukkan bahwa semakin banyak jumlah iterasinya, maka semakin baik pula performa BER-nya.

Kode turbo tersusun dari dua atau lebih komponen kode dan komponen *interleaver*. Komponen enkodernya sendiri merupakan konvolusional enkoder. Oleh karena itu, komponen dekodernya harus mampu menangani transfer informasi dari satu dekoder ke dekoder lain (*priori information*). Maka,komponen dekodernya harus menggunak-

an teknik *soft decision decoding* (SDD) agar didapatkan hasil yang optimal. Hal ini kontras dengan *hard decision decoding* (HDD). Contoh dari komponen dekoder ini adalah SOVA [**?**] dan algoritma MAP [**?**]. Pada subbab di bawah akan dibahas sekilas mengenai konvolusional enkoder, turbo enkoder, dan turbo dekoder.

2.1.1. Konvolusional Enkoder

Konvolusional enkoder dapat dibagi menjadi dua, yaitu sistematik dan non sistematik. Sistematik konvolusional enkoder berarti urutan data masuk menjadi bagian dari *codeword*, sehingga dapat langsung dikenali pada keluaran enkoder. *Codeword* adalah beberapa urutan keluaran enkoder yang telah dilakukan proses *multiplexing* menjadi hanya satu urutan data keluaran. Struktur turbo enkoder akan dijelaskan lebih jauh setelah parameter konvolusional enkoder dijabarkan terlebih dahulu.

Coding rate konvolusional enkoder didefinisikan sebagai $R = k/n$, dengan k-bit masukan dan n-bit keluaran. Selain parameter tersebut, konvolusional enkoder juga memiliki parameter orde memori dan *constraint length*. Orde memori m didefinisikan sebagai

$$m = \underset{1 \leq i \leq k}{max} v_i; \tag{2.1}$$

dengan m adalah panjang dari semua k *shift register*. Sedangkan *constraint length* didefinisikan sebagai

$$v = \sum_{1 \leq i \leq k} v_i; \tag{2.2}$$

dengan v adalah jumlah dari semua panjang k *shift register*. Pada umumnya konvolusional enkoder penyebutan parameter dalam pendefinisian konvolusional enkoder adalah (n, k, v) enkoder. Selain itu, konvolusional enkoder juga memiliki parameter *generator polynomial* g yang menggambarkan koneksi terhadar register.

Codeword yang dihasilkan konvolusional enkoder merupakan operasi konvolusi masukan dengan *generator polynomial* konvolusional enkoder. *Codeword* juga dapat dihasilkan dengan metode *Finite State Machine* (FSM) atau *state space* karena pada dasarnya konvolusional enkoder merupakan rangkaian linear *shift registers*.

2.1.2. Turbo Enkoder

Struktur umum turbo enkoder ditunjukkan seperti pada gambar 2.2. Dua komponen konvolusional enkoder digunakan untuk melakukan proses *encoding*, tetapi sebuah *in-*

Gambar 2.2: Struktur umum turbo enkoder [**?**]

Gambar 2.3: Struktur umum turbo dekoder [**?**]

terleaver ditempatkan diantara keduanya. Jenis konvolusional enkoder yang digunakan pada struktur ini adalah *Recursive Systematic Code* (RSC) yaitu konvolusional enkoder yang memiliki umpan balik. Selain itu, turbo enkoder dapat memiliki lebih dari dua komponen enkoder.

Pada struktur turbo enkoder terdapat dua jenis informasi, yaitu sistematik informasi (informasi langsung dari masukan) dan informasi *parity* (hasil dari konvolusional enkoder). Kemudian kedua informasi inilah yang telah melewati proses *multiplexing* disebut dengan *codeword*.

Interleaver sendiri adalah metode untuk memisahkan masukan simbol dalam domain waktu. Pemisahan ini dilakukan dengan cara memetakan satu-satu urutan x menjadi urutan simbol $\tilde{x}$ dengan fungsi pemetaan $\lambda(t)$, seperti yang didefinisikan oleh persamaan berikut

$$\tilde{x}_t = x_{\lambda(t)}, \;\; t \in [0, K-1], \tag{2.3}$$

dengan K adalah panjang masukan simbol.

Di komunitas peneliti, *interleaver* telah banyak diterima bahwa *interleaver* memiliki peranan yang sangat penting dalam perancangan kode turbo yang baik [**?**]. Pada paper asli kode turbo sendiri [**?**] juga dinyatakan bahwa dengan bertambah panjangnya ukuran *interleaver* dapat meningkatkan performa kode turbo.

2.1.3. Turbo Dekoder

Tujuan dari turbo dekoder adalah merekonstruksi ulang data yang telah ditransmisikan oleh pengirim, baik data tersebut rusak karena derau ataupun tidak. Struktur umum turbo dekoder ditunjukkan seperti pada gambar 2.3. Struktur turbo dekoder ini memiliki dua komponen dekoder yang dihubungkan oleh blok *interleaver*. Seperti yang terlihat pada gambar masing-masing dekoder menerima 3 masukan, yaitu keluaran sistematik enkoder, *parity* bit yang ditransmisikan dari komponen enkoder yang bersesuaian dan informasi dari komponen dekoder lain. Informasi dari dekoder lain ini selanjutnya akan disebut dengan *priori information*.

Gambar 2.4: Struktur turbo enkoder sesuari standar 3GPP [**?**]

Komponen dekoder ini harus dapat menangani *priori information* yang berupa SDD. Selain itu dikarenakan turbo dekoder ini bersifat iteratif maka komponen dekoder inipun juga harus dapat menghasilkan keluaran SDD untuk komponen dekoder lain. SDD ini pada umumnya direpresentasikan sebagai *Log Likelihood Ratio* (LLR). Karena pada tugas akhir ini dibatasi dengan penggunakan BPSK sebagai teknik modulasinya, maka HDD cukup memeriksa polaritas dari LLR ini. Komponen dekoder yang mendukung transfer informasi SDD ini adalah SOVA dan MAP.

Turbo dekoder pada gambar 2.3 beroperasi secara iteratif dan pada iterasi pertama komponen dekoder pertama menerima nilai keluaran saja dan menghasilkan LLR untuk komponen dekoder kedua. Kemudian LLR ini digunakan sebagai tambahan informasi bagi komponen dekoder kedua. Lalu iterasi kedua dapat dimulai dengan komponen turbo dekodernya melakukan proses *decoding* nilai keluaran ditambah dengan LLR dari komponen dekoder kedua pada iterasi sebelumnya. Tambahan informasi ini menyebabkan tingkat akurasi *decoding* lebih tinggi dan hasilnya digunakan oleh komponen dekoder kedua. Proses ini berulang dengan tiap iterasinya BER dari bit yang mengalami proses *decoding* cenderung turun. Namun, karena bersifat iteratif, perlu diperhatikan bahwa pada blok *interleaver* tidak menggunakan ulang informasi yang sama lebih dari sekali di tiap proses *decoding*.

2.2. Kode Turbo Standar 3GPP

Pada subbab ini akan dijabarkan mengenai kode turbo sesuai standar 3GPP [**?**]. Pada standar hanya dijelaskan mengenai spesifikasi turbo enkoder lengkap dengan spesifikasi *interleaver*. Oleh karena itu pada subbab ini, khusus untuk turbo dekoder akan dijelaskan pada bab selanjutnya.

2.2.1. Turbo Enkoder

Struktur turbo enkoder yang digunakan pada standar adalah *Parallel Concatenated Convolutional Code* (PCCC) dengan dua 8-*state* komponen enkoder dan satu internal *interleaver*. *Coding rate* turbo enkoder ini adalah $1/3$. Gambar 2.4 mengillustrasikan spesifikasi yang ada pada standar.

Generator polynomial dari 8-*state* PCCC adalah sebagai berikut

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right], \tag{2.4}$$

dimana

$$g_0(D) = 1 + D^2 + D^3,$$
$$g_1(D) = 1 + D + D^3.$$

Nilai awal dari semua *shift register* adalah nol.

Keluaran dari turbo enkoder ini adalah

$$\begin{aligned} d_K^{(0)} &= x_K \\ d_K^{(1)} &= z_K \\ d_K^{(2)} &= z'_K \end{aligned} \tag{2.5}$$

untuk $k = 0, 1, \ldots, K-1$, dengan K adalah jumlah bit masukan.

Bit masukan turbo enkoder dinotasikan dengan $c_0, c_1, \ldots, c_{K-1}$ dan bit keluaran dari komponen enkoder pertama dinotasikan dengan $z_0, z_1, \ldots, z_{K-1}$ dan untuk komponen enkoder kedua dinotasikan dengan $z'_0, z'_1, \ldots, z'_{K-1}$. Sedangkan bit keluaran dari internal *interleaver* turbo enkoder dinotasikan dengan $c'_0, c'_1, \ldots, c'_{K-1}$ dan bit ini merupakan masukan dari komponen enkoder kedua.

2.2.2. Teralis Terminasi untuk Turbo Enkoder

Teralis terminasi dilakukan dengan cara mengambil bit terakhir dari *shift register* setelah semua informasi bit telah dilakukan proses *encoding*. Tiga bit pertama dari bit terakhir digunakan untuk menghasilkan tiga bit *tail* komponen enkoder (saklar beralih ke posisi bawah).

Persamaan bit *tail*-nya adalah sebagai berikut

$$\begin{aligned} d_K^{(0)} &= x_K, & d_{K+1}^{(0)} &= z_{K+1}, \\ d_K^{(1)} &= z_K, & d_{K+1}^{(1)} &= x_{K+2}, \\ d_K^{(2)} &= x_{K+1}, & d_{K+1}^{(2)} &= z_{K+2}, \\ \\ d_{K+2}^{(0)} &= x'_K, & d_{K+3}^{(0)} &= z'_{K+1}, \\ d_{K+2}^{(1)} &= z'_K, & d_{K+3}^{(1)} &= x'_{K+2}, \\ d_{K+2}^{(2)} &= x'_{K+1}, & d_{K+3}^{(2)} &= z'_{K+2}. \end{aligned} \tag{2.6}$$

2.2.3. Internal *Interleaver* Turbo Enkoder

Hubungan antara bit masukan dengan bit keluaran blok ini adalah sebagai berikut

$$c'_i = c_{\pi(i)}, i = 0, 1, \ldots, K-1, \tag{2.7}$$

$$\pi(i) = (f_1.i + f_2.i^2)\ mod\ (K). \tag{2.8}$$

Parameter f_1 dan f_2 bergantung pada ukuran blok K dan terangkum pada tabel 2.1.

Tabel 2.1: Tabel parameter f_1 dan f_2 terhadap K

i	K	f_1	f_2	i	K	f_1	f_2	i	K	f_1	f_2	i	K	f_1	f_2
1	40	3	10	**48**	416	25	52	**95**	1120	67	140	**142**	3200	111	240
2	48	7	12	**49**	424	51	106	**96**	1152	35	72	**143**	3264	443	204
3	56	19	42	**50**	432	47	72	**97**	1184	19	74	**144**	3328	51	104
4	64	7	16	**51**	440	91	110	**98**	1216	39	76	**145**	3392	51	212
5	72	7	18	**52**	448	29	168	**99**	1248	19	78	**146**	3456	451	192
6	80	11	20	**53**	456	29	114	**100**	1280	199	240	**147**	520	257	220
7	88	5	22	**54**	464	247	58	**101**	1312	21	82	**148**	3584	57	336
8	96	11	24	**55**	472	29	118	**102**	1344	211	252	**149**	3648	313	228
9	104	7	26	**56**	480	89	180	**103**	1376	21	86	**150**	3712	271	232
10	112	41	84	**57**	488	91	122	**104**	1408	43	88	**151**	3776	179	236
11	120	103	90	**58**	496	157	62	**105**	1440	149	60	**152**	3840	331	120
12	128	15	32	**59**	504	55	84	**106**	1472	45	92	**153**	3904	363	244
13	136	9	34	**60**	512	31	64	**107**	1504	49	846	**154**	3968	375	248
14	144	17	108	**61**	528	17	66	**108**	1536	71	48	**155**	4032	127	168
15	152	9	38	**62**	544	35	68	**109**	1568	13	28	**156**	4096	31	64
16	160	21	120	**63**	560	227	420	**110**	1600	17	80	**157**	4160	33	130
17	168	101	84	**64**	576	65	96	**111**	1632	25	102	**158**	4224	43	264
18	176	21	44	**65**	592	19	74	**112**	1664	183	104	**159**	4288	33	134
19	184	57	46	**66**	608	37	76	**113**	1696	55	954	**160**	4352	477	408
20	192	23	48	**67**	624	41	234	**114**	1728	127	96	**161**	4416	35	138
21	200	13	50	**68**	640	39	80	**115**	1760	27	110	**162**	4480	233	280
22	208	27	52	**69**	656	185	82	**116**	1792	29	112	**163**	4544	357	142
23	216	11	36	**70**	672	43	252	**117**	1824	29	114	**164**	4608	337	480
24	224	27	56	**71**	688	21	86	**118**	1856	57	116	**165**	4672	37	146
25	232	85	58	**72**	704	155	44	**119**	1888	45	354	**166**	4736	71	444
26	240	29	60	**73**	720	79	120	**120**	1920	31	120	**167**	4800	71	120
27	248	33	62	**74**	736	139	92	**121**	1952	59	610	**168**	4864	37	152
28	256	15	32	**75**	752	23	94	**122**	1984	185	124	**169**	4928	39	462
29	264	17	198	**76**	768	217	48	**123**	2016	113	420	**170**	4992	127	234
30	272	33	68	**77**	784	25	98	**124**	2048	31	64	**171**	5056	39	158
31	280	103	210	**78**	800	17	80	**125**	2112	17	66	**172**	5120	39	80
32	288	19	36	**79**	816	127	102	**126**	2176	171	136	**173**	5184	31	96
33	296	19	74	**80**	832	25	52	**127**	2240	209	420	**174**	5248	113	902
34	304	37	76	**81**	848	239	106	**128**	2304	253	216	**175**	5312	41	166
35	312	19	78	**82**	864	17	48	**129**	2368	367	444	**176**	5376	251	336
36	320	21	120	**83**	880	137	110	**130**	2432	265	456	**177**	5440	43	170
37	328	21	82	**84**	896	215	112	**131**	2496	181	468	**178**	5504	21	86
38	336	115	84	**85**	912	29	114	**132**	2560	39	80	**179**	5568	43	174
39	344	193	86	**86**	928	15	58	**133**	2624	27	164	**180**	5632	45	176
40	352	21	44	**87**	944	147	118	**134**	2688	127	504	**181**	5696	45	178
41	360	133	90	**88**	960	29	60	**135**	2752	143	172	**182**	5760	161	120
42	368	81	46	**89**	976	59	122	**136**	2816	43	88	**183**	5824	89	182
43	376	45	94	**90**	992	65	124	**137**	2880	29	300	**184**	5888	323	184
44	384	23	48	**91**	1008	55	84	**138**	2944	45	92	**185**	5952	47	186
45	392	243	98	**92**	1024	31	64	**139**	3008	157	188	**186**	6016	23	94
46	400	151	40	**93**	1056	17	66	**140**	3072	47	96	**187**	6080	47	190
47	408	155	102	**94**	1088	171	204	**141**	3136	13	28	**188**	6144	263	48

Gambar 3.1: Kompleksitas sistem dengan 4 macam level abstraksi [**?**]

BAB 3

PEMODELAN FUNGSIONAL RTL MENGGUNAKAN SYSTEMC

Pada bab sebelumnya telah dijelaskan mengenai teori kode turbo, maka pada dua bab selanjutnya akan dijelaskan mengenai teori metode perancangan yang akan dilakukan yakni pemodelan fungsional RTL dengan systemC dan ABV. Namun pada bab ini akan dibahas mengenai penggunaan systemC sebagai *tool* pemodelan fungsional RTL. Selain akan dibahas mengenai fitur-fitur dan model perhitungan dengan tingkat abstraksi yang berbeda-beda, pada bab ini akan dijelaskan terlebih dahulu mengenai tujuan penggunaan systemC pada tugas akhir ini.

3.1. Tujuan Penggunaan SystemC

Kecenderungan perancangan sistem yang telah bergeser [**?**][**?**] menjadi motivasi utama penggunaan systemC sebagai *tool* pemodelan yang baru. Pada umumnya sistem yang ada sekarang ini terdiri dari HW, misalnya *Application-Specification Integrated Circuit* (ASIC) dan SW. Oleh karena itu, pada umumnya HW dan SW dikembangkan bersamaan (*co-developed*) dengan jadwal yang sangat ketat dan dengan spesifikasi yang kompleks. Istilah yang sering digunakan untuk pendekatan perancangan secara bersamaan dan multi disiplin ini adalah ESL.

Teknik ESL ini merupakan tanggapan dari permasalah tingkat kompleksitas sistem yang semakin tinggi dengan waktu perancangan yang semakin pendek. Akibatnya banyak pertimbangan untung-rugi terhadap optimasi HW, SW, dan performa sistem secara keseluruhan dengan melibatkan multi disiplin ilmu. Utamanya ESL ini fokus pada pengembangan HW, SW, dan algoritma. Gambar 3.1 menjelaskan kompleksitas perancangan HW pada *System on Chip* (SOC).

Selain dari segi kompleksitas yang semakin bertambah, semakin pendeknya waktu perancangan sistem juga merupakan isu utama saat ini. Alasan diperpendeknya waktu

perancangan dikarenakan melihat kebutuhan siklus produksi yang semakin cepat di pasaran. Selain itu, gap yang jauh antara bahasa pemograman pemodelan dengan HW juga menjadi kendala dengan kondisi waktu perancangan yang pendek [**?**]. Oleh karena itu, dibutuhkan tambahan sumber daya, termasuk sumber daya manusia, *tool*, dll.

Untuk mendukung kedua isu tersebut (kompleksitas sistem dan waktu perancangan yang semakin pendek) dibutuhkan *tool* yang mendukung proses pengembangan HW dan SW secara bersamaan. SystemC mendukung proses tersebut. Selain fitur tersebut, systemC juga mendukung tingkat pemodelan sistem dalam beberapa tingkatan abstraksi dan sampai saai ini pun systemC ini masih dikembangkan agar dapat langsung disintesis tanpa melalui tahap perancangan HW menggunakan HDL. Namun pada tugas akhir kali ini, tidak akan dilakukan pemodelan dengan systemC yang dapat disintesis karena masih belum ada *tool* yang mendukung sehingga didapatkan HW yang optimum [**?**].

Sebagai kesimpulannya, motivasi utama penggunaan systemC adalah menyediakan kerangka pemodelan untuk sistem dimana model fungsional tingkat tinggi dapat diubah ke tingkat yang lebih rendah secara halus hanya dalam satu bahasa.

3.2. Sekilas Mengenai SystemC

SystemC adalah *library* C++ yang dikembangkan oleh OSCI Language Working Group[1]. SystemC menyediakan beberapa kelas khusus, tipe data, dan makro yang digunakan untuk berbagai macam komponen model. Pada subbab kali ini akan dibahas secara singkat fitur-fitur systemC yang disediakan.

3.2.1. `Modules` dan `Processes`

Blok dasar dari systemC adalah `modules` dan `processes`. Blok `modules` digunakan untuk memecah sistem yang kompleks dan dapat digunakan secara hirarki. Blok `modules` juga menggunakan ports untuk komunikasi dan dapat memiliki variable internal. Cara pendefinisian `modules` yaitu dengan menggunakan makro `SC_MODULE`.

Bagian fungsional dari sebuah `modules` diimplementasikan dengan menggunakan `processes`. Pada dasarnya `processes` ini dijalankan secara bersamaan. Blok `processes` ini juga dibagi menjadi dua tipe berbeda, yaitu `method` dan `thread`.

[1]Open SystemC Initiative (OSCI), diakses pada tanggal 21 Juli 2011; http://www.systemc.org.

Gambar 3.2: Model Perhitungan SystemC [**?**]

Perbedaan utama dari kedua tipe ini adalah `method` akan selalu mengeksekusi program dari awal hingga akhir tanpa adanya interupsi, sedangkan `thread` memiliki kemampuan menunda dan melanjutkannya. Pendefinisiannya menggunakan `SC_METHOD` dan `SC_THREAD`. Masing-masing `module` tersebut memiliki konstruktor yang memetakan anggota fungsinya. Cara pendefinisiannya menggunakan `SC_CTOR`.

Salah satu contoh tipe data port dalam module tersebut adalah `sc_in` dan `sc_out` untuk mendefinisikan port masukan dan keluaran. Selain itu masih terdapat banyak tipe port lain yang mendukung ESL, misalnya `sc_fifo` yang memodelkan antarmuka *First-in First-out* (FIFO). Pemodelan *fixed point* pun dapat dilakukan dengan menggunakan systemC. Hal ini merupakan fitur unggulan dari systemC sebagai bahasa pemodelan. Cara pendefinisiannya dengan menggunakan `SC_FIX`.

3.3. Model Perhitungan SystemC

Pada bahasan sebelumnya telah dijelaskan mengenai berbagai level abstraksi pemodelan yang disediakan oleh systemC. Berdasar [**?**] model perhitungan yang disediakan systemC dapat dirangkum menjadi lima, yaitu *an untimed functional model, a timed functional model, a transaction-level model, a behavioral hardware model*, dan *a register-transfer model* seperti tampak pada gambar 3.2.

3.3.1. Model Fungsional *Untimed*

Model fungsional *untimed* adalah spesifikasi fungsional dari sistem yang akan dibuat. Semua fungsional sistem diimplementasikan di model ini, namun tidak ada referensi tentang detil arsitektur sistem yang akan dibuat. Sesuai dengan namanya tingkat abstraksi ini tidak memiliki informasi waktu. Ketika model ini disimulasikan, hanya hasil fungsionalnya yang dapat diverifikasi. Model ini juga sering disebut dengan spesifikasi *executeble*.

3.3.2. Model Fungsional *Timed*

Sama dengan model sebelumnya, model ini memiliki fungsionalistas yang sama, namun model ini memiliki tingkat akurasi perkiraan waktu yang lebih tinggi. Aspek lain selain aspek akurasi waktu ini sama dengan model *untimed*.

3.3.3. Model *Transaction-Level*

Model ini mendefinisikan komunikasi antar modul dengan menggunakan pemanggilan fungsi. Peningkatan akurasi pada model ini terletak pada protokol komunikasinya. Oleh karena itu, terdapat isolasi antara blok komunikasi dengan blok perhitungannya. Model *transaction-level* ini memiliki keterangan pendekatan waktu baik protokol komunikasinya maupun modul perhitungannya untuk mengindikasikan perkiraaan kasar karakteristik waktu sistem. Pada model ini, modul merepresentasikan komponen perhitungan sedangkan pemanggilan fungsi merepresentasikan protokol komunikasi.

3.3.4. Model *Behavior Hardware*

Model *behavior hardware* tidak memiliki isolasi antara blok komunikasi dan blok perhitungannya yang berarti protokol komunikasinya diselipkan elemen perhitungannya. Dibanding komunikasi antarmuka bus yang digunakan seperti pada model *transaction-level*, model ini menggunakan `wires` sebagai konektor antar portnya. Pada model ini, tingkat akurasi waktunya masih perkiraan.

3.3.5. Model *Register Transfer Level*

Model ini adalah model yang paling akurat yang didukung oleh systemC. Sehingga model ini memiliki gap bahasa yang dekat dengan HDL. Semua komunikasi, perhitungan, dan aspek arsitektur dari target sistem didefinisikan secara eksplisit. Karakteristik waktu baik perhitungan dan komunikasi memiliki akurasi berdasarkan *clock*. Pemodelan inilah yang akan digunakan pada tugas akhir ini.

3.4. Aturan Translasi SystemC ke HDL (Verilog)

Pada tugas akhir ini, karena tidak menggunakan *tool* untuk mengubah model systemC menjadi HDL (verilog), maka proses pengubahan ini dilakukan secara manual. Oleh karena itu, pada bagian ini akan dijelaskan beberapa aturan yang dilakukan pada proses translasi systemC ke HDL (verilog) yang dilakukan pada tugas akhir ini.

3.4.1. Deklarasi Port dan Variabel Internal

Secara umum proses translasi untuk deklarasi port atau tipe data terdiri dari empat variasi, yaitu deklarasi port masukan dan keluaran, deklarasi tipe data `sc_uint` atau

`sc_int`, dan deklarasi tipe data *fixed point*. Tabel 3.1 menyimpulkan proses translasi yang dilakukan.

Program 3.1: Deklarasi Port dan Variabel Internal pada SystemC

```
sc_in    <sc_fixed<WORD,INT,SC_RND_MIN_INF,SC_SAT> >  code1_8k;
sc_in    <bool>                                       first_it_beta;
sc_out   <sc_uint<10> >                               out;
         sc_fixed<BETA,INTBETA,SC_RND_MIN_INF,SC_SAT> beta1_8k;
 ...
```

Tabel 3.1: Tabel translasi deklarasi port dan variabel internal systemC ke HDL (verilog)

SystemC	HDL (Verilog)	keterangan
sc_in	input wire/reg	deklarasi port
sc_out	output wire/reg	deklarasi port
bool	tidak perlu deklarasi panjang data	pada verilog cukup dideklarasikan port
sc_uint<PANJANG>	[(PANJANG-1):0]	berlaku juga untuk tipe data sc_int
sc_fixed<WORD,...>	[(WORD-1):0]	berlaku juga untuk tipe data turunannya

Program 3.2: Deklarasi Port dan Variabel Internal pada Verilog

```
input   wire    [(WORD-1):0] code1_8k,
input   wire                 first_it_beta,
output  wire    [9:0]        out,
        reg     [(BETA-1):0] beta1_8k;
 ...
```

3.4.2. Pemanggilan Modul

Terdapat perbedaan pemanggilan modul pada systemC dan verilog. Jika pada systemC deklarasi nama *instance* modul dan port terpisah, maka pada verilog tidak terpisah. Berikut adalah contoh translasi pemanggilan modul.

Program 3.3: Pemanggilan Modul pada SystemC

```
betacomp_comb comb;

SC_CTOR(betacomp) : comb("COMB")
  {

        comb.code1_8k(code1_8k);
        comb.code1_8k1(code1_8k1);
        comb.code1_8k2(code1_8k2);
```

```
        comb.code1_8k3(code1_8k3);
        comb.code1_8k4();
        comb.code1_8k5(code1_8k5);
        comb.code1_8k6(code1_8k6);
        comb.code1_8k7(code1_8k7);
...
```

Program 3.4: Pemanggilan Modul pada Verilog

```
betacomp_comb comb(
        .code1_8k(code1_8k),
        .code1_8k1(code1_8k1),
        .code1_8k2(code1_8k2),
        .code1_8k3(code1_8k3),
        .code1_8k4(),
        .code1_8k5(code1_8k5),
        .code1_8k6(code1_8k6),
        .code1_8k7(code1_8k7),
...
```

3.4.3. Deklarasi *Sensitivity List*

Pada systemC deklarasi *sensitivity list* terletak pada bagian konstruktor, sedangkan pada verilog terletak pada bagian blok `always`. Berikut adalah contoh proses translasinya.

Program 3.5: Deklarasi *Sensitivity List* pada SystemC

```
SC_METHOD (comb);
        sensitive  << code1_8k;
        sensitive  << code1_8k1;
        sensitive  << code1_8k2;
        sensitive  << code1_8k3;
...
```

Program 3.6: Deklarasi *Sensitivity List* pada Verilog

```
always @(
        code1_8k  or
        code1_8k1 or
        code1_8k2 or
        code1_8k3
...
```

BAB 4

VERIFIKASI BERBASIS *ASSERTION*

Pada bab ini akan dijelaskan mengenai metode verifikasi HW yang digunakan yaitu ABV. Sebelum dijelaskan detil mengenai metode ini, akan dijelaskan terlebih dahulu mengenai verifikasi secara umum.

4.1. Pendahuluan

Setiap tahap perancangan sebuah chip membutuhkan verifikasi untuk memastikan bahwa setiap tahapan tersebut telah sesuai dengan yang diharapkan. Sehingga suksesnya aspek verifikasi akan berimbas pada suksesnya keseluruhan tahap proses perancangan chip dan akan menjamin kualitas sesuai dengan spesifikasi yang didefinisikan di awal perancangan.

Dalam verifikasi, aktivitas yang paling banyak dilakukan adalah proses *debugging*, yaitu proses mencari *bug* dan mengoreksinya. Menurut penelitian yang dilakukan oleh FarWest Research tahun 2007, sebesar 52% dari keseluruhan aktivitas verifikasi dihabiskan untuk debugging (lihat gambarv4.1). Proses *debugging* merupakan proses yang sangat sulit dan kompleks karena terkadang tidak hanya berhubungan dengan tahap yang sedang diverifikasi semata.

Gambar 4.1: Persentase aktivitas pada proses verifikasi

Metode verifikasi konvensional memiliki berbagai kelemahan [**?**], misalnya dengan kompleksitas desain yang semakin tinggi, maka diperlukan *test vector* yang semakin banyak pula. Hal ini membutuhkan konsumsi waktu dan tenaga yang banyak serta, sangat tidak efisien, terlebih ketika waktu perancangan yang sangat ketat. Kelemahan selanjutnya adalah tidak dapat menemukan kesalahan dengan lebih cepat dan tepat karena mekanisme yang dipakai adalah pembandingan data tanpa adanya informasi dimana dan kapan kesalahan tersebut terjadi. Selain itu, metode verifikasi konvensional

Gambar 4.2: *Platform* ABV

menggunakan pendekatan verifikasi *black box*, dimana dalam melakukan verifikasi tidak meninjau sinyal-sinyal yang ada di dalam desain. Hal ini terkadang menyebabkan kejadian dimana sinyal di dalam desain tidak sesuai spesifikasi, namun menghasilkan keluaran yang benar. Hal ini dikarenakan metode ini tidak memiliki tingkat observabilitas yang tinggi.

Kelemahan-kelemahan tersebut dapat teratasi dengan adanya metode ABV. Metode ini menggabungkan teknik *assertion-based methodology* dan *property checking*. Properti adalah pernyataan tentang sifat yang diharapkan. Sedangkan metode *assertion* cara mengimplementasikan properti tersebut. Dengan metode ini desain yang telah dibuat lebih dapat dipastikan kebenarannya dibandingkan metode konvensional. *Platform* ABV, terdiri dari *constrained random* sebagai metode pemberian stimulus, *transaction-based* untuk menghubungkan HW dan SW, *assertion-based* untuk memeriksa properti/spesifikasi desain, dan *functional coverage* untuk mengukur keterselesaian/keberhasilan proses verifikasi. *Platform* ABV dapat dilihat pada gambar 4.2.

Namun, pada tahap pemodelan fungsional (bukan menggunakan systemC) akan digunakan skenario *average test* dan *corner test* [**?**]. Skenario tes ini dimotivasi ketika proses verifikasi dihadapkan pada skenario tes yang banyak. Oleh karena itu dipilih beberapa tes yang dapat mewakili semua kemungkinan tes tersebut. *Corner test* mewaliki skenario tes untuk nilai-nilai ekstrim, misalnya nilai parameter paling rendah atau paling tinggi, pergantian nilai parameter, dll. Sedangkan *average case* mewakili skenario umum dan skenario ini dilakukan berulang kali hingga mencapai tingkat keyakinan tertentu. Pada bab ini tidak akan dibahas khusus mengenai metode ini karena merupakan metode verifikasi umum yang digunakan pada tahapan pemodelan fungsional.

Selanjutnya akan dijelaskan lebih rinci mengenai *platform* ABV seperti pada gambar 4.2.

4.2. Stimulus *Constrained Random*

Jika pada *directed test*, *bug* yang ditemukan adalah *bug* yang sudah diprediksi ada di dalam desain, namun pada *constrained random test*, *bug* yang tidak pernah terprediksi tetap dapat ditemukan. Saat melakukan pengetesan *constrained random*, diperlukan

functional coverage untuk mengetahui sejauh mana perkembangan verifikasi. Dengan *constrained random* tidak akan banyak waktu terbuang untuk menulis *test vector*. *Constrained random testbench* yang dibuat dapat digunakan berulang-ulang untuk skenario tes yang berbeda. Kekurangannya hanya untuk membuat *testbench* tersebut diperlukan waktu yang lebih lama dibandingkan directed *testbench* sehingga akan ada penundaan di awal. Setelah itu, praktis akan banyak waktu yang dapat dihemat.

Penggunaan *constrained random* dapat menjangkau area verifikasi yang lebih luas yang tidak dapat dijangkau oleh *directed test*. Untuk dapat menjangkau kondisi-kondisi khusus, dapat ditambahkan kriteria *constraint* pada *testbench* dan disimulasikan berkali-kali sehingga ditemukan *bug*.

4.3. *Assertion Checking*

Pada ABV dapat diverifikasi benar tidaknya protokol yang telah dirancang. Pengecekan protokol berfokus pada sinyal kontrol. Dalam verifikasi sebuah desain, validitas sinyal kontrol memegang peranan yang sangat penting. Selain protokol, metode ini dapat juga membantu dalam proses *debugging* dengan cara membuat properti untuk tipe data tertentu atau port data tertentu.

Dua istilah yang dominan dalam metode ini adalah properti dan *assertion*. Properti adalah sebuah pernyataan tentang sifat yang diharapkan. Sedangkan *assertion* sebuah implementasi dari properti yang dievaluasi dan dieksekusi menggunakan *tool* untuk memvalidasi spesifikasi desain.

Menurut [**?**], dinyatakan bahwa pengecekan properti pada ABV dapat dilakukan dengan dua jalan:

1. Jika properti yang seharusnya terjadi saat simulasi, namun kenyataannya tidak terjadi, maka *assertion* gagal.
2. Jika properti yang tidak boleh terjadi saat simulasi, namun kenyataannya terjadi, maka *assertion* gagal.

Foster, Krolnik, dan Lacey [**?**] menjelaskan bahwa penggunaan metode verifikasi berbasis assertion memberikan banyak manfaat, di antaranya:

1. Meningkatkan observabilitas
   Dengan menggunakan metode berbasis *assertion*, observabilitas akan meningkat

yang berarti verifikasi tidak lagi bergantung pada propagasi dari input. Meningkatnya observabilitas akan menghasilkan keuntungan sebagai berikut:

- Meningkatkan deteksi kesalahan
- Meningkatkan notifikasi kesalahan

2. Mengurangi waktu yang diperlukan untuk proses *debugging*
   ABV memungkinkan tim verifikasi menemukan kapan dan dimana lokasi kesalahan terjadi. Sehingga hal tersebut dapat mengurangi waktu proses *debugging* dan mempercepat produksi suatu chip.

3. Meningkatkan efisiensi verifikasi
   Seperti dijelaskan pada poin sebelumnya, ABV memungkinkan tim verifikasi menemukan secara cepat dan tepat kapan dan dimana kesalahan terjadi. Sehingga hal tersebut dapat meningkatkan efisiensi waktu proses *debugging* secara signifikan.

4. Meningkatkan kemudahan komunikasi melalui dokumentasi
   Dengan menambahkan *assertion*, tim lain dapat dengan mudah memahami properti atau spesifikasi suatu desain.

4.4. *Functional Coverage*

Functional coverage berfungsi untuk mengukur seberapa banyak desain implementasi telah sesuai dengan spesifikasi awalnya. Ukuran pada *functional coverage* disebut *coverage points*. *Coverage points* merupakan sampling dari sebuah ekspresi. Tujuannya adalah untuk memastikan bahwa semua ekspresi yang relevan telah diperiksa pada saat ekspresi tersebut disampling. Oleh karena itu, sebelum diimplementasikan perlu diperjelas lagi ekspresi apa yang akan disampling, dimana dan kapannya. Selain itu, penting juga untuk didefinisikan apa saja yang membuat ekspresi tersebut relevan. *Coverage points* dapat juga digunakan untuk mendeteksi kesalahan seperti statemen `if` pada kode *testbench*. Namun, kesalahan yang dideteksi lebih pada sinyal kontrol atau protokol, bukan pada kebenaran data keluaran. Pengecekan terhadap kebenaran data keluaran dilakukan pada dengan cara membandingkan dengan keluaran data dari model. Implementasi metrik *functional coverage* dinyatakan benar jika mencapai 100% yang berarti bahwa semua coverage point yang ada di simulasi telah terpenuhi, dengan kata lain fungsional desain implementasi telah sesuai dengan spesifikasi desain yang tertuang dalam model functional coverage.

BAB 5

OPTIMASI PERANCANGAN HARDWARE KODE TURBO

Bab ini akan memaparkan teknik atau algoritma dasar yang akan digunakan sebagai landasan dalam mengoptimasi kekurangan perancangan kode turbo menjadi HW. Kemudian, setelah itu akan dipaparkan teknik atau algoritma optimasi yang dilakukan. Namun pada awalnya, perlu dijabarkan mengenai kekurangan-kekurangan jika kode turbo ini diimplementasi menjadi HW yang menjadi alasan mengapa perlu dilakukan optimasi.

5.1. Kekurangan Implementasi HW Kode Turbo

Pada turbo enkoder, kekurangan utamanya jika akan diimplementasikan menjadi HW adalah adanya penundaan pada blok *interleaver* sebelum bit masukan diolah oleh komponen enkodernya. Penundaan waktu ini diakibatkan oleh blok *interleaver* yang membutuhkan seluruh data tersedia terlebih dahulu. Selain itu dari sisi kompleksitas perhitungannya pun seperti yang ditunjukkan pada persamaan 2.8 (indeks *generator*) dan tabel 2.1 memiliki operator dengan *critical path* panjang yakni 1 multiplier (13 bit $\times$ 13 bit) (panjang bit i dikali panjang bit i) dan 1 multiplier (26 bit $\times$ 10 bit) (2 kali panjang bit i dikali dengan panjang bit maksimum f_2 saat $K = 5248$ (lihat tabel 2.1)).

Untuk memperpendek waktu latensi pemrosesan turbo enkoder, maka diperlukan teknik paralelisasi. Namun seiring dengan bertambahnya level paralelisasi maka bertambah pula kerugian yang diakibatkan seperti bertambah besarnya area arsitektur dan dalam beberapa kasus panjang *critical path* bertambah pula. Selain itu paralelisasi juga harus dilakukan untuk mengatasi masalah penundaan waktu antara blok *interleaver* dengan komponen enkoder.

Sedangkan untuk turbo dekoder, panjang latensinya pun yang tinggi karena bersifat iteratif. Selain itu, telah disebutkan pada bab sebelumnya bahwa komponen dekodernya memiliki tingkat kompleksitas perhitungan yang tinggi. Pada struktur seperti tampak

pada gambar 2.3 terdapat dua blok *interleaver* yang tentunya menambah panjang latensi.

Pemilihan komponen dekoder pada perancangan turbo dekoder ini juga penting karena performa BER yang dihasilkan sangat bergantung pada komponen dekoder. Blok *interleaver* yang digunakan identik dengan *interleaver* yang digunakan di turbo enkoder.

Setelah dirangkum mengenai kekurangan-kekurangan pada perancangan HW kode turbo, maka pada subbab selanjutnya akan dijelaskan algoritma MAP yang dipilih sebagai komponen dekoder pada turbo dekoder.

5.2. Algoritma MAP

Seperti yang telah disampaikan secara sekilas pada bab sebelumnya mengenai penggunaan LLR sebagai jenis informasi antar komponen dekoder, maka pada subbab ini alangkah lebih baiknya jika konsep mengenai LLR ini dijabarkan terlebih dahulu.

5.2.1. *Log Likelihood Ratios*

Konsep LLR ini pada awalnya diperkenalkan oleh Robertson [**?**] untuk menyederhanakan informasi antar komponen dekoder.

LLR dari bit input u_k yang bergantung pada bit *parity* atau selanjutnya akan disebutkan dengan istilah peluang *posteriori* didefinisikan sebagai berikut dasar pendekatan dari algoritma Max-log-MAP.

$$L(u_k|y) \triangleq ln\left(\frac{P(u_k = +1|y)}{P(u_k = -1|y)}\right). \tag{5.1}$$

Pada persamaan di atas, perlu diperhatikan bahwa nilai $+1$ merupakan menunjukkan bit 1 sedangkan nilai -1 menunjukkan bit 0. Untuk model kanal dengan modulasi BPSK, maka besar dari nilai LLR ini mengindikasikan kemungkinan nilai u_k yang benar. Misalnya jika $L(u_k) \gg 0$, maka kemungkinan besar nilai bitnya adalah 1.

Setelah memperkenalkan konsep LLR, maka pada bahasan selanjutnya akan dijabarkan mengenai algoritma MAP yang akan digunakan sebagai dasar pendekatan yang dilakukan algoritma Max-log-MAP.

Gambar 5.1: Kemungkinan transisi turbo enkoder [**?**]

5.2.2. Algoritma *Maximum A-Posteriori*

Pada tahun 1974, Bahl, Cocke, Jelinek, dan Raviv memperkenalkan algoritma MAP [**?**][**?**] yang dapat mengestimasi peluang *posteriori* dari diagram *trellis*. Algoritma ini juga dikenal dengan nama algoritma BCJR [**?**] yang merupakan singkatan dari huruf depan keempat penemu algoritma tersebut. Mereka menunjukkan bagaimana algoritma mereka tidak hanya sanggup mengestimasi urutan bit, namun juga dapat mengestimasi peluang untuk masing-masing bit apakah sudah dilakukan proses *decoding* dengan benar. Hal ini sangat penting untuk proses *decoding* yang bersifat iteratif seperti kode turbo.

Dasar dari perhitungan matematis algoritma MAP adalah aturan Bayes sebagai berikut

$$P(a \wedge b) = P(a|b) \cdot P(b). \tag{5.2}$$

Konsekuensi dari aturan Bayes di atas adalah

$$P(a \wedge b|c) = P(a|b \wedge c) \cdot P(b|c), \tag{5.3}$$

yang dapat diturunkan dari persamaan 5.2 dengan $x \equiv a \wedge b$ dan $y \equiv b \wedge c$ seperti berikut

$$\begin{aligned} P(a \wedge b|c) \equiv P(x|c) \qquad & = \frac{P(x \wedge c)}{P(c)} \\ = \frac{P(a \wedge b \wedge c)}{P(c)} \qquad & \equiv \frac{P(a \wedge y)}{P(c)} \\ = \frac{P(a|y) \cdot P(y)}{P(c)} \qquad & \equiv P(a|b \wedge c) \cdot \frac{P(a \wedge y)}{P(c)} \\ = P(a|b \wedge c) \cdot P(b|c). & \end{aligned} \tag{5.4}$$

Dengan menggunakan persamaan 5.2, maka persamaan 5.1 dapat ditulis ulang menjadi

$$L(u_k|y) \triangleq ln\left(\frac{P(u_k = +1 \wedge y)}{P(u_k = -1 \wedge y)}\right). \tag{5.5}$$

Pada gambar 5.1 menunjukkan kemungkinan transisi komponen enkoder (lihat gam-

bar 2.4). Jika *state* sebelumnya merupakan S_{k-1} dan *state* sekarang S_k diketahui, maka nilai u_k yang menyebabkan transisi *state* tersebut diketahui juga. Transisi tersebut memiliki sifat *mutually exclusive* yang berarti bahwa hanya satu transisi yang dapat terjadi di satu waktu. Oleh karena itu, peluang salah satu transisi tersebut terjadi merupakan penjumlahan peluang individualnya. Sehingga persamaan 5.5 dapat ditulis menjadi

$$L(u_k|y) \triangleq ln \left(\frac{\sum_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_k=+1} P(\mathbf{S_{k-1}} = \acute{\mathbf{s}} \wedge \mathbf{S_{k-1}} = \mathbf{s} \wedge \mathbf{y})}{\sum_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_k=-1} P(\mathbf{S_{k-1}} = \acute{\mathbf{s}} \wedge \mathbf{S_{k-1}} = \mathbf{s} \wedge \mathbf{y})} \right). \tag{5.6}$$

Urutan data y dapat dipecah menjadi tiga bagian, yaitu *codeword* yang berhubungan dengan transisi *state* sekarang y_k, *state* sebelumnya $y_{j<k}$ dan *state* selanjutnya $y_{j>k}$. Oleh karena itu, didapatkan persamaan sebagai berikut

$$P(\acute{s} \wedge s \wedge y) = P(\acute{s} \wedge s \wedge y_{j<k} \wedge y_k \wedge y_{j>k}). \tag{5.7}$$

Dengan menggunakan persamaan 5.2 dan dengan diasumsikan bahwa kanalnya *memoryless* sehingga nilai $y_{j>k}$ hanya bergantung kepada *state* sekarang s, maka didapatkan persamaan sebagai berikut

$$\begin{aligned} P(\acute{s} \wedge s \wedge y) &= P(y_{j>k}|\acute{s} \wedge s \wedge y_{j<k} \wedge y_k) \cdot P(\acute{s} \wedge s \wedge y_{j<k} \wedge y_k) \\ &= P(y_{j>k}|s) \cdot P(\acute{s} \wedge s \wedge y_{j<k} \wedge y_k). \end{aligned} \tag{5.8}$$

Sekali lagi dengan menggunakan aturan yang sama persamaan sebelumnya dapat dikembangkan lebih jauh menjadi

$$\begin{aligned} P(\acute{s} \wedge s \wedge y) &= P(y_{j>k}|s) \cdot P(\acute{s} \wedge s \wedge y_{j<k} \wedge y_k) \\ &= P(y_{j>k}|s) \cdot P(y_k \wedge s|\acute{s} \wedge y_{j<k}) \cdot P(\acute{s} \wedge y_{j<k}) \\ &= P(y_{j>k}|s) \cdot P(y_k \wedge s|\acute{s}) \cdot P(\acute{s} \wedge y_{j<k}) \\ &= \beta_k(s) \cdot \gamma_k(\acute{s}, s) \cdot \alpha_{k-1}, \end{aligned} \tag{5.9}$$

dimana

$$\begin{aligned} \alpha_{k-1}(\acute{s}) &= P(S_{k-1} = \acute{s} \wedge y_{j<k}), \\ \beta_k(s) &= P(y_{j>k}|S_k = s), \\ \gamma_k(\acute{s}, s) &= P(\{y_k \wedge S_k = s\}|S_{k-1} = \acute{s}). \end{aligned} \tag{5.10}$$

Gambar 5.2: Trellis MAP dekoder

Untuk selanjutnya, $\alpha_{k-1}(\acute{s})$ akan disebut dengan *forward recursion metric*, $\beta_k(s)$ akan disebut dengan *backward recursion metric*, dan $\gamma_k(\acute{s}, s)$ akan disebut dengan *branch metric*. Gambar 5.2 mengillustrasikan penurunan persamaan di atas.

Dengan mensubtitusi persamaan 5.9 ke persamaan 5.6, maka didapatkan persamaan di bawah.

$$L(u_k|y) \triangleq ln \left(\frac{\sum\limits_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_k=+1} \beta_k(s) \cdot \gamma_k(\acute{s}, s) \cdot \alpha_{k-1}(\acute{s})}{\sum\limits_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_k=-1} \beta_k(s) \cdot \gamma_k(\acute{s}, s) \cdot \alpha_{k-1}(\acute{s})} \right). \tag{5.11}$$

Implementasi HW operator logaritma natural seperti yang ditunjukkan persamaan di atas akan menjadi kompleks. Oleh karena itu pada bahasan selanjutnya akan dipaparkan algoritma Max-log-MAP yang melakukan pendekatan dalam implementasi algoritma MAP.

5.3. Algoritma Max-log-MAP

Dasar dari algoritma Max-log-MAP adalah pendekatan logaritma Jacobian seperti yang pertama kali digunakan oleh Erfanian [**?**][**?**]. Persamaan di bawah merupakan pendekatan yang digunakan algoritma Max-log-MAP ini.

$$ln \left(\sum_{\mathbf{i}} e^{x_i} \right) \approx \max_i(x_i). \tag{5.12}$$

Langkah pertama algoritma Max-log-MAP ini adalah mendeklarasikan ulang persamaan 5.11 menjadi

$$L(u_k|y) \triangleq ln \left(\frac{\sum\limits_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_k=+1} exp(A_{k-1}(\acute{s}) + B_k(s) + \Gamma_k(\acute{s}, s))}{\sum\limits_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_k=-1} exp(A_{k-1}(\acute{s}) + B_k(s) + \Gamma_k(\acute{s}, s))} \right) \tag{5.13}$$

, dengan $A_{k-1}(\acute{s}) \triangleq ln(\alpha_{k-1})$, $B_k(s) \triangleq ln(\beta_k(s))$, dan $\Gamma_k(\acute{s}, s) \triangleq ln(\gamma_k(\acute{s}, s))$.

Gambar 5.3: Diagram algoritma Max-log-MAP

Sehingga dengan mensubtitusi persamaan 5.12 pada persamaan 5.13 akan didapatkan persamaan akhir dalam algoritma Max-log-MAP di bawah.

$$L(u_k|y) \approx \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_k=+1}(A_{k-1}(\acute{s}) + B_k(s) + \Gamma_k(\acute{s}, s)) - \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_k=-1}(A_{k-1}(\acute{s}) + B_k(s) + \Gamma_k(\acute{s}, s)). \tag{5.14}$$

Hal ini berarti bahwa algoritma ini menghitung *posteriori* LLR $L(u_k|y)$ untuk masing-masing bit u_k dengan mempertimbangkan setiap transisi dari S_{k-1} ke S_k yang mengelompokkan grup antara kejadian nilai $u_k = +1$ dan $u_k = -1$. Untuk kedua grup ini, transisi memberi nilai maksimum dari $A_{k-1}(\acute{s}),\ B_k(s),\ \Gamma_k(\acute{s}, s)$. *Posteriori* LLR dihitung hanya dengan berdasakan nilai maksimum tersebut. Gambar 5.3 merangkum alur pengerjaan algoritma Max-log-MAP.

5.3.1. Perhitungan *Metric* pada Algoritma Max-log-MAP

Pada bagian ini akan dijelaskan lebih rinci mengenai perhitungan masing-masing *metric*. Pada tugas akhir kali ini, arsitektur acuan yang dipakai tampak seperti pada gambar 5.4.

Gambar 5.4: Arsitektur Turbo Dekoder [**?**]

Branch Metric

Branch metric berhubungan dengan cabang yang menghubungkan titik *state* sebelum dan *state* sekarang. Persamaan umum dari *branch metric* dengan notasi yang dipakai pada gambar 5.4 adalah $\Gamma_{ij} = V(X_k)X(i,j) + R(Z_k)Z(i,j)$, dimana $X(i,j)$ adalah bit masukan yang berhubungan dengan transisi yang terjadi, sedangkan $Z(i,j)$ adalah bit *parity* yang berhubungan dengan transisi yang terjadi. Oleh karena itu, persamaan

branch metric-nya menjadi

$$\begin{aligned} BM_0 &= 0 \\ BM_1 &= V(X_k) \\ BM_2 &= R(Z_k) \\ BM_3 &= V(X_k) + R(Z_k) \end{aligned} \tag{5.15}$$

$$\Gamma_{ij} = BM(f(i,j))$$

$$f(i,j) = \begin{cases} 0 & ,i = 0\ dan\ j = 0 \\ 0 & ,i = 1\ dan\ j = 0 \\ 1 & ,i = 2\ dan\ j = 0 \\ 1 & ,i = 3\ dan\ j = 0 \\ 1 & ,i = 4\ dan\ j = 0 \\ 1 & ,i = 5\ dan\ j = 0 \\ 0 & ,i = 6\ dan\ j = 0 \\ 0 & ,i = 7\ dan\ j = 0 \\ 3 & ,i = 0\ dan\ j = 1 \\ 3 & ,i = 1\ dan\ j = 1 \\ 2 & ,i = 2\ dan\ j = 1 \\ 2 & ,i = 3\ dan\ j = 1 \\ 2 & ,i = 4\ dan\ j = 1 \\ 2 & ,i = 5\ dan\ j = 1 \\ 3 & ,i = 6\ dan\ j = 1 \\ 3 & ,i = 7\ dan\ j = 1 \end{cases}$$

$$i = 0, 1, \ldots, 7$$

$$j = 0, 1$$

, dengan komponen dekoder pertama $V(X_k) = V_1(X_k)$ dan untuk dekoder kedua adalah $V(X_k) = V_2(X_k)$ dan $R(Z_k) = R(Z'_k)$. Gambar 5.5 menunjukkan kemungkinan transisi pada kode turbo sesuai dengan standar dan lengkap dengan *branch metric* yang

bersesuaian.

Gambar 5.5: Kemungkinan transisi turbo enkoder dengan *branch metric* yang bersesuaian

Backward Recursion Metric

Notasi *backward recursion metric* untuk *state* ke i S_i pada teralis ke k adalah $B_k(S_i)$, dengan $2 \leq k \leq K+3$ dan $0 \leq i \leq 7$. *Metric* ini diinisialisasi dengan $B_{K+3}(S_0) = 0$ dan $B_{K+3}(S_i) = -\infty, i > 0$. Untuk teralis ke $k = K+2$ sampai $k = 2$, nilai *backward recursion metric* dapat dicari dengan persamaan

$$B_k(S_i) = max\{(B_{k+1}(S_{j0}) + \Gamma_{ij0}), (B_{k+1}(S_{j1}) + \Gamma_{ij1})\}. \tag{5.16}$$

Notasi j yang diikuti dengan angka menunjukkan perubahan *state* akibat bit angka tersebut.

Forward Recursion Metric

Notasi *forward recursion metric* untuk *state* ke i S_i pada teralis ke k adalah $A_k(S_i)$, dengan $0 \leq k \leq K-1$ dan $0 \leq i \leq 7$. *Metric* ini diinisialisasi dengan $A_0(S_0) = 0$ dan $A_0(S_i) = -\infty, i > 0$. Untuk teralis ke $k = 1$ sampai $k = K$, nilai *forward recursion metric* dapat dicari dengan persamaan

$$A_k(S_i) = max\{(A_{k-1}(S_{j0}) + \Gamma_{ij0}), (A_{k-1}(S_{j1}) + \Gamma_{ij1})\}. \tag{5.17}$$

Notasi j yang diikuti dengan angka menunjukkan perubahan *state* akibat bit angka tersebut.

5.4. Algoritma *Sliding Window*

Seperti yang dapat dilihat pada gambar 5.3, sekilas dapat dibayangkan bahwa implementasi algoritma Max-log-MAP membutuhkan jumlah memori yang besar untuk menyimpan hasil perhitungan *metric*-nya dan membutuhkan waktu proses yang lama

untuk menyelesaikan proses *decoding*-nya. Oleh karena itu, untuk mengatasi kekurangan tersebut dapat digunakan SWA. Pada dasarnya algoritma ini memotong perhitungan *forward recursion metric* atau *backward recursion metric* sehingga didapat jumlah memori yang lebih kecil dan waktu latensi yang lebih pendek. Namun, algoritma ini menyebabkan performa kode turbo berkurang. Agar kerugian performa ini berkurang, maka digunakan *guard window* untuk menoleransinya. Semakin panjang *guard window* ini, maka semakin mengecil pula kerugian performanya.

5.5. Optimasi Turbo Enkoder

Pada subbab ini akan dijabarkan mengenai optimasi yang dilakukan pada blok turbo enkoder, baik blok konvolusional enkoder dan blok *interleaver*. Turbo enkoder ini dapat dibagi menjadi tiga bagian, yaitu konvolusional enkoder, teralis terminasi, dan blok *interleaver*.

5.5.1. Konvolusional Enkoder dan Teralis Terminasinya

Dengan menggunakan model matematis *state space* maka struktur konvolusional enkoder (komponen enkoder tanpa teralis terminasi) yang ditunjukkan pada gambar 5.6 memiliki persamaan sebagai berikut

Gambar 5.6: Konvolusional enkoder

$$\begin{aligned} X_{t+1} &= FX_t + GU_t \\ Y_t &= HX_t + JU_t \end{aligned} \tag{5.18}$$

$$X_t = \begin{pmatrix} X_t^1 \\ X_t^2 \\ X_t^3 \end{pmatrix}$$

$$F = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, G = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$H = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}, J = \begin{pmatrix} 1 \end{pmatrix}$$

Untuk mendapatkan latensi yang lebih pendek maka diajukan perancangan turbo enkoder yang memiliki tingkat paralelisasi 8 yang berarti dalam satu waktu mengolah 8 bit masukan langsung dan menghasilkan 8 bit keluaran pula. Dalam kasus model *state space* ini berarti perhitungan *state* selanjutnya juga harus memiliki 8 level paralelisasi. Persamaan umum *next state* ke-n adalah sebagai berikut

$$X_{t+n} = FX_{t+(n-1)} + GU_{t+(n-1)}, \{n \in \mathbb{Z}^+ | n > 0\} \tag{5.19}$$

Agar didapatkan hubungan X_{t+n} dengan X_t, maka $X_{t+(n-1)}$ perlu dijabarkan sehingga persamaan 5.19 menjadi seperti di bawah.

$$X_{t+n} = F^n X_t + F^{n-1} GU_t + F^{n-2} GU_{t+1} + \cdots + GU_{t+(n-1)} \tag{5.20}$$

Sedangkan untuk persamaan keluaran *state* adalah sebagai berikut.

$$Y_{t+n} = HX_{t+(n-1)} + JU_{t+(n-1)} \tag{5.21}$$

Sehingga dengan menggunakan persamaan 5.20 dan 5.21 didapatkan persamaan *state* 8 level paralelisasinya adalah sebagai berikut.

$$\begin{aligned} X_{t+8} = {} & F^8 X_t + F^7 GU_t + F^6 GU_{t+1} + F^5 GU_{t+2} + F^4 GU_{t+3} + \\ & F^3 GU_{t+4} + F^2 GU_{t+5} + FGU_{t+6} + GU_{t+7} \end{aligned} \tag{5.22}$$

$$
\begin{aligned}
Y_t &= HX_t + JU_t \\
Y_{t+1} &= HFX_t + HGU_t + JU_{t+1} \\
Y_{t+2} &= HF^2X_t + HFGU_t + HGU_{t+1} + JU_{t+2} \\
Y_{t+3} &= HF^3X_t + HF^2GU_t + HFGU_{t+1} + HGU_{t+2} + JU_{t+3} \\
Y_{t+4} &= HF^4X_t + HF^3GU_t + HF^2GU_{t+1} + HFGU_{t+2} + HGU_{t+3} + \\
&\quad JU_{t+4} \\
Y_{t+5} &= HF^5X_t + HF^4GU_t + HF^3GU_{t+1} + HF^2GU_{t+2} + HFGU_{t+3} + \\
&\quad HGU_{t+4} + JU_{t+5} \\
Y_{t+6} &= HF^6X_t + HF^5GU_t + HF^4GU_{t+1} + HF^3GU_{t+2} + HF^2GU_{t+3} + \\
&\quad HFGU_{t+4} + HGU_{t+5} + JU_{t+6} \\
Y_{t+7} &= HF^7X_t + HF^6GU_t + HF^5GU_{t+1} + HF^4GU_{t+2} + HF^3GU_{t+3} + \\
&\quad HF^2GU_{t+4} + HFGU_{t+5} + HGU_{t+6} + JU_{t+7}
\end{aligned}
\tag{5.23}
$$

Selanjutnya dengan mensubtitusikan nilai perkalian matriks antara **H**, **F**, **G**, dan **J**, untuk $i = 0, 1, \ldots, \frac{K}{8} - 1$, dimana $t = 8i$, maka didapat persamaan 5.24.

$$
\begin{aligned}
X^1_{(8i)+8} &= X^2_{(8i)} + X^3_{(8i)} + U_{(8i)} + U_{(8i)+3} + U_{(8i)+4} + U_{(8i)+5} + U_{(8i)+7} \\
X^2_{(8i)+8} &= X^1_{(8i)} + U_{(8i)+2} + U_{(8i)+3} + U_{(8i)+4} + U_{(8i)+6} \\
X^3_{(8i)+8} &= X^2_{(8i)} + U_{(8i)+1} + U_{(8i)+2} + U_{(8i)+3} + U_{(8i)+5}
\end{aligned}
\tag{5.24}
$$

$$
\begin{aligned}
Y_{(8i)} &= X^1_{(8i)} + X^2_{(8i)} + U_{(8i)} \\
Y_{(8i)+1} &= X^1_{(8i)} + X^2_{(8i)} + X^3_{(8i)} + U_{(8i)} + U_{(8i)+1} \\
Y_{(8i)+2} &= X^1_{(8i)} + X^3_{(8i)} + U_{(8i)} + U_{(8i)+1} + U_{(8i)+2} \\
Y_{(8i)+3} &= X^3_{(8i)} + U_{(8i)} + U_{(8i)+1} + U_{(8i)+2} + U_{(8i)+3} \\
Y_{(8i)+4} &= X^2_{(8i)} + U_{(8i)+1} + U_{(8i)+2} + U_{(8i)+3} + U_{(8i)+4} \\
Y_{(8i)+5} &= X^1_{(8i)} + U_{(8i)+2} + U_{(8i)+3} + U_{(8i)+4} + U_{(8i)+5} \\
Y_{(8i)+6} &= X^2_{(8i)} + X^3_{(8i)} + U_{(8i)} + U_{(8i)+3} + U_{(8i)+4} + U_{(8i)+5} + U_{(8i)+6} \\
Y_{(8i)+7} &= X^1_{(8i)} + X^2_{(8i)} + U_{(8i)+1} + U_{(8i)+4} + U_{(8i)+5} + U_{(8i)+6} + U_{(8i)+7}
\end{aligned}
$$

Gambar 5.7: Tabel implementasi teknik *common term reduction*

$$i = 0, 1, \ldots, \frac{K}{8} - 1$$

Namun, persamaan di atas dapat dioptimasi dengan teknik yang disebut dengan *common term reduction* [**?**]. Pada dasarnya teknik ini menemukan operan yang sama dan menyimpan hasilnya untuk sementara waktu. Hasil dari teknik ini ditunjukkan seperti pada gambar 5.7.

Dengan menggunakan teknik *common term reduction*, maka didapat persamaan akhir 8-level konvolusional enkoder yang telah teroptimasi sebagai berikut.

$$\begin{aligned}
S_1 &= X^1_{(8i)} + X^2_{(8i)}, P_1 = X^2_{(8i)} + S_2 \\
S_2 &= X^3_{(8i)} + U_{(8i)}, P_2 = X^2_{(8i)} + S_3 \\
S_3 &= U_{(8i)+1} + U_{(8i)+2}, P_3 = U_{(8i)+1} + S_1 \\
S_4 &= X^1_{(8i)} + X^2_{(8i)}, P_4 = U_{(8i)+5} + S_5 \\
S_5 &= U_{(8i)+3} + U_{(8i)+4}
\end{aligned} \tag{5.25}$$

$$\begin{aligned}
Q_1 &= S_2 + S_3 \\
Q_2 &= P_1 + P_4
\end{aligned}$$

$$\begin{aligned}
X^1_{(8i)+8} &= Q_2 + U_{(8i)+7} \\
X^2_{(8i)+8} &= S_4 + S_5 + U_{(8i)+6} \\
X^3_{(8i)+8} &= P_2 + U_{(8i)+3} + U_{(8i)+5}
\end{aligned}$$

$$\begin{aligned}
Y_{(8i)} &= S_1 + U_{(8i)} \\
Y_{(8i)+1} &= P_3 + S_2 \\
Y_{(8i)+2} &= X^1_{(8i)} + Q_1 \\
Y_{(8i)+3} &= Q_1 + U_{(8i)+3} \\
Y_{(8i)+4} &= P_2 + S_5
\end{aligned}$$

$$
\begin{aligned}
Y_{(8i)+5} &= S_4 \quad + \; P_4 \\
Y_{(8i)+6} &= Q_2 \quad + \; U_{(8i)+6} \\
Y_{(8i)+7} &= P_3 \quad + \; U_{(8i)+4} \; + \; U_{(8i)+5} \; + \; U_{(8i)+6} \; + \; U_{(8i)+7}
\end{aligned}
$$

$$
i = 0, 1, \ldots, \frac{K}{8} - 1
$$

$$
\begin{aligned}
X_0^1 &= 0 \\
X_0^2 &= 0 \\
X_0^3 &= 0
\end{aligned}
$$

Sedangkan untuk penentuan keluaran hasil teralis terminasinya tidak diperlukan penurunan matematis. Hal ini dikarenakan urutan *state*-nya tidak bergantung pada bit masukan, sehingga urutannya tetap untuk masing-masing kemungkinan *state*. Keluaran untuk teralis terminasi ditunjukkan oleh table 5.1.

Tabel 5.1: Tabel keluaran teralis terminasi

$\mathbf{X^1_{(8i)+8}}$	$\mathbf{X^2_{(8i)+8}}$	$\mathbf{X^3_{(8i)+8}}$	$\mathbf{x_K}$ atau $\mathbf{x'_K}$	$\mathbf{x_{K+1}}$ atau $\mathbf{x'_{K+1}}$	$\mathbf{x_{K+2}}$ atau $\mathbf{x'_{K+2}}$	$\mathbf{z_K}$ atau $\mathbf{z'_K}$	$\mathbf{z_{K+1}}$ atau $\mathbf{z'_{K+1}}$	$\mathbf{z_{K+2}}$ atau $\mathbf{z'_{K+2}}$
0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0	0
0	1	0	1	1	0	0	1	0
0	1	1	0	1	0	1	1	0
1	0	0	0	1	1	1	0	1
1	0	1	1	1	1	0	0	1
1	1	0	1	0	1	1	1	1
1	1	1	0	0	1	0	1	1

Gambar 5.8 mengillustrasikan arsitektur komponen enkoder yang telah dioptimasi (persamaan 5.25 dan tabel 5.1). Blok `ROM` merupakan implementasi dari tabel 5.1. Gambar tersebut disusun 4 *stage* yang menunjukkan *critical path*-nya.

Gambar 5.8: Arsitektur komponen enkoder yang telah dioptimasi (persamaan 5.25 dan tabel 5.1)

5.5.2. Indeks *Generator* pada Blok *Interleaver*

Langkah awal optimasi pada blok ini sama dengan optimasi pada blok turbo enkoder yaitu dengan cara paralelisasi persamaan awal menjadi 8-level. Persamaan 2.8 dengan versi 8-level paralelnya ditunjukkan seperti pada persamaan di bawah.

$$\begin{aligned}
\pi(8j) &= \begin{cases} 0 & ,j = 0 \\ (f_1(8j) + f_2(8j)^2)\ mod\ (K) & ,j \neq 0 \end{cases} \\
\pi(8j+1) &= (f_1(8j+1) + f_2(8j+1)^2)\ mod\ (K) \\
\pi(8j+2) &= (f_1(8j+2) + f_2(8j+2)^2)\ mod\ (K) \\
\pi(8j+3) &= (f_1(8j+3) + f_2(8j+3)^2)\ mod\ (K) \\
\pi(8j+4) &= (f_1(8j+4) + f_2(8j+4)^2)\ mod\ (K) \\
\pi(8j+5) &= (f_1(8j+5) + f_2(8j+5)^2)\ mod\ (K) \\
\pi(8j+6) &= (f_1(8j+6) + f_2(8j+6)^2)\ mod\ (K) \\
\pi(8j+7) &= (f_1(8j+7) + f_2(8j+7)^2)\ mod\ (K)
\end{aligned} \tag{5.26}$$

$$j = 0, 1, \ldots, \frac{K}{8} - 1.$$

Persamaan 5.26 memiliki jumlah operator perkalian yang banyak akibat proses paralelisasi. Selain itu, *critical path*-nya pun bertambah panjang. Oleh karena itu, langkah selanjutnya adalah mengurangi jumlah perkalian dengan membuat nilai indeks bergantung dengan indeks berikutnya. Pembuktian 1 menunjukkan kebergantungan antar nilai indeks. Dengan menggunakan 2 definisi di bawah definisi (1 dan definisi 2), maka akan dibuktikan secara bertahap pembuktian 1.

Definition 1 *r = a mod d,* $\{a \in \mathbb{Z}, d \in \mathbb{Z}^+\}$*, then there are unique integers q and r, with* $0 \leq r < d$ *such that a = dg + r.*

Definition 2 *r = a mod d, then r = a,* $\{a, d \in \mathbb{Z}^+ | a < d\}$

Proof 1 $\pi(i+1) = (f_1 + f_2 + \pi(i) + 2f_2 i)\ mod\ (K), \{f_1, f_2, K \in \mathbb{Z}^+ | f_1 + f_2 < K\}$

$$\pi(i) = (f_1 i + f_2 i^2)\ mod\ (K)$$

$$\equiv \langle i = j + 1, \{j \in \mathbb{N} | j = 0, 1, \ldots, (K-1)\}\rangle$$

$$\pi(j+1) = (f_1(j+1) + f_2(j+1)^2)\ mod\ (K)$$

$$\equiv \langle (j+1)^2 = j^2 + 2j + 1\rangle$$

$$\pi(j+1) = (f_1(j+1) + f_2(j^2 + 2j + 1))\ mod\ (K)$$

$$\equiv \langle Distributivity\rangle$$

$$\pi(j+1) = (f_1 j + f_1 + f_2 j^2 + 2f_2 j + f_2)\ mod\ (K)$$

$$\equiv \langle Associativity\rangle$$

$$\pi(j+1) = (f_1 + f_2 + f_1 j + f_2 j^2 + 2f_2 j)\ mod\ (K)$$

$$\equiv \langle Definition\ 1\ over\ Eq\ 2.8,\ i = j, f_1 j + f_2 j^2 = \pi(j) + Kq\rangle$$

$$\pi(j+1) = (f_1 + f_2 + Kq + \pi(j) + 2f_2 j)\ mod\ (K)$$

$$\equiv \langle Definition\ 2,\ a = f1 + f2, d = K\rangle$$

$$\pi(j+1) = (f_1 + f_2 + \pi(j) + 2f_2 j)\ mod\ (K),$$

$$\{f_1, f_2, K \in \mathbb{Z}^+ | f_1 + f_2 < K\}$$

$$\equiv \langle j = i\rangle$$

$$\pi(i+1) = (f_1 + f_2 + \pi(i) + 2f_2 i)\ \ mod\ \ (K),$$

$$\{f_1, f_2, K \in \mathbb{Z}^+ | f_1 + f_2 < K\}$$

Pembuktian ini valid dan dapat diiimplementasikan karena syarat $\{f_1, f_2, K \in \mathbb{Z}^+ | f_1 + f_2 < K\}$ terpenuhi untuk semua kemungkinan nilai f_1 dan f_2 seperti yang tampak pada tabel 2.1. Oleh karena itu, persamaan ini dapat diperluas menjadi 8-level paralelisasi

sehingga menjadi persamaan di bawah.

$$\begin{aligned}\gamma_1 &= f_1 + f_2 \\ \gamma_2 &= 2f_2\end{aligned}$$

$$\begin{aligned}
X_1 &= \gamma_1 + \gamma_2.8j \quad , F_1 = \pi(8j) + X_1 \\
Y_1 &= X_1 + \gamma_2 \quad , F_2 = F_1 + Y_1 \\
Y_2 &= Y_1 + \gamma_2 \quad , F_3 = F_2 + Y_2 \\
Y_3 &= Y_2 + \gamma_2 \quad , F_4 = F_3 + Y_3 \\
Y_4 &= Y_3 + \gamma_2 \quad , F_5 = F_4 + Y_4 \\
Y_5 &= Y_4 + \gamma_2 \quad , F_6 = F_5 + Y_5 \\
Y_6 &= Y_5 + \gamma_2 \quad , F_7 = F_6 + Y_6 \\
Y_7 &= Y_6 + \gamma_2 \quad , F_8 = F_7 + Y_7
\end{aligned} \tag{5.27}$$

$$\pi(8j) = \begin{cases} 0 & , j = 0 \\ F_8 \; mod \; (K) & , j \neq 0 \end{cases}$$

$$\begin{aligned}
\pi(8j+1) &= F_1 \; mod \; (K) \\
\pi(8j+2) &= F_2 \; mod \; (K) \\
\pi(8j+3) &= F_3 \; mod \; (K) \\
\pi(8j+4) &= F_4 \; mod \; (K) \\
\pi(8j+5) &= F_5 \; mod \; (K) \\
\pi(8j+6) &= F_6 \; mod \; (K) \\
\pi(8j+7) &= F_7 \; mod \; (K)
\end{aligned}$$

, dengan syarat

$$\{f_1, f_2, K \in \mathbb{Z}^+ | f_1 + f_2 < K\}$$

, dimana

$$\begin{aligned}
j &= 0, 1, \ldots, \frac{K}{8} - 1 \\
i &= 8j
\end{aligned}$$

Gambar 5.9 mengillustrasikan arsitektur implementasi HW persamaan 5.27. Jalur yang ditandai warna oranye merupakan *critical path* baik untuk turbo enkoder maupun turbo dekoder. Hal ini akan dijelaskan lebih rinci pada bab 8.

Gambar 5.9: Arsitektur indeks *generator* pada blok *interleaver* sesuai persamaan 5.27

5.5.3. Top Level Arsitektur Turbo Enkoder

Top level arsitektur yang ditunjukkan pada gambar 2.4 dioptimasi dengan menggunakan *dual bank ram* agak panjang latensi menjadi semakin pendek. Sehingga jika menggunakan persamaan yang telah dioptimasi dan *dual bank ram* ini, maka dibanding turbo enkoder yang asli, arsitektur ini lebih cepat maksimal hingga 16 kali.

Kecepatan ini mungkin dicapai jika ukuran urutan data yang masuk K sama dan urutan datanya yang diolah tersebut banyak. Jika nilai ukuran data tersebut dinyatakan dengan K_x, maka waktu yang dibutuhkan untuk mengolah n urutan data masuk adalah T_{opt} untuk arsitektur yang dioptimasi dan T_{nopt} untuk arsitektur yang tidak dioptimasi. Arsitektur turbo enkoder yang dioptimasi memiliki maksimal hingga R_{opt} kali lebih cepat.

$$T_{opt} = \frac{1}{8} \times \sum_{i=1}^{n+1} K_x = \frac{1}{8} \times (n+1) \times K_x. \tag{5.28}$$

$$T_{nopt} = 2 \times \sum_{i=1}^{n} K_x = 2 \times n \times K_x. \tag{5.29}$$

$$R_{opt} = \lim_{n \to \infty} \frac{T_{nopt}}{T_{opt}} = 16. \tag{5.30}$$

Gambar 5.10 menunjukkan top level arsitektur yang telah di optimasi. Blok `PROPOSED INTERLEAVER` mengacu kepada gambar 5.9. Sedangkan, blok `PROPOSED CONSTITUENT ENCODER` mengacu pada gambar 5.8.

Gambar 5.10: Top level arsitektur turbo enkoder yang dioptimasi

5.6. Optimasi Turbo Dekoder

Setelah dijelaskan mengenai optimasi yang dilakukan pada blok turbo enkoder, maka pada kali ini akan dijelaskan mengenai optimasi yang dilakukan pada turbo dekoder, yakni implementasi SWA, paralelisasi pada komponen dekodernya, dan penggunaan

ulang blok dekoder sehingga didapatkan arsitektur yang lebih efisien dibanding arsitektur aslinya [**?**].

5.6.1. Implementasi SWA

Pada tugas akhir ini digunakan *window* SWA dengan panjang tetap yaitu 40 yang mana merupakan ukuran *window* terkecil yang ditunjukkan oleh tabel 2.1. Panjang *guard window* yang digunakan adalah 8. Hal ini dipilih karena turbo dekoder yang dirancang sudah memiliki hasil performa BER yang lebih baik dibanding turbo dekoder acuan (lihat bab 6 atau subbab 6.1.2 atau gambar 6.6).

Gambar 5.11: Diagram waktu implementasi SWA

Gambar 5.11 menunjukkan alur implementasi SWA. Garis putus-putus mengillustrasikan perhitungan *backward recursion metric* dan sekaligus *branch metric*. Sedangkan, garis sambung mengillustrasikan perhitungan *forward recursion metric*, *branch metric*, dan *posteriori* LLR sekaligus. Daerah dengan berarsir abu-abu mengillustrasikan perhitungan *guard window*. Semua perhitungan ini dilakukan secara paralel 8-level.

5.6.2. Paralelisasi Komponen Dekoder

Optimasi perhitungan *metric* secara paralel ini hanya dilakukan pada *backward recursion metric* dan *forward recursion metric* karena perhitungan *branch metric* dilakukan bersamaan dengan kedua *metric* tersebut.

Perhitungan 8-Level *Backward Recursion Metric*

Untuk perhitungan teralis terminasinya menggunakan persamaan di bawah.

$$\begin{aligned} B_{K+2}(S_i) &= max\{(B_{K+3}(S_{j0}) + \Gamma_{ij0}), (B_{K+3}(S_{j1}) + \Gamma_{ij1})\} \\ B_{K+1}(S_i) &= max\{(B_{K+2}(S_{j0}) + \Gamma_{ij0}), (B_{K+2}(S_{j1}) + \Gamma_{ij1})\} \\ B_{K}(S_i) &= max\{(B_{K+1}(S_{j0}) + \Gamma_{ij0}), (B_{K+1}(S_{j1}) + \Gamma_{ij1})\} \end{aligned} \tag{5.31}$$

, dengan nilai inisialisasi $B_{K+3}(S_0) = 0$ dan $B_{K+3}(S_i) = -\infty, i > 0$.

Sedangkan untuk $z = \frac{K}{8} - 1, \ldots, 1, 0$ menggunakan persamaan sebagai berikut.

$$\begin{aligned}
B_{(8\times(z+1))}(S_i) &= max\{\big(B_{(8\times(z+1))+1}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(B_{(8\times(z+1))+1}(S_{j1}) + \Gamma_{ij1}\big)\} \\
B_{(8\times(z+1))-1}(S_i) &= max\{\big(B_{(8\times(z+1))} \quad (S_{j0}) + \Gamma_{ij0}\big) \quad , \big(B_{(8\times(z+1))} \quad (S_{j1}) + \Gamma_{ij1}\big)\} \\
B_{(8\times(z+1))-2}(S_i) &= max\{\big(B_{(8\times(z+1))-1}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(B_{(8\times(z+1))-1}(S_{j1}) + \Gamma_{ij1}\big)\} \\
B_{(8\times(z+1))-3}(S_i) &= max\{\big(B_{(8\times(z+1))-2}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(B_{(8\times(z+1))-2}(S_{j1}) + \Gamma_{ij1}\big)\} \\
B_{(8\times(z+1))-4}(S_i) &= max\{\big(B_{(8\times(z+1))-3}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(B_{(8\times(z+1))-3}(S_{j1}) + \Gamma_{ij1}\big)\} \\
B_{(8\times(z+1))-5}(S_i) &= max\{\big(B_{(8\times(z+1))-4}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(B_{(8\times(z+1))-4}(S_{j1}) + \Gamma_{ij1}\big)\} \\
B_{(8\times(z+1))-6}(S_i) &= max\{\big(B_{(8\times(z+1))-5}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(B_{(8\times(z+1))-5}(S_{j1}) + \Gamma_{ij1}\big)\} \\
B_{(8\times(z+1))-7}(S_i) &= max\{\big(B_{(8\times(z+1))-6}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(B_{(8\times(z+1))-6}(S_{j1}) + \Gamma_{ij1}\big)\}
\end{aligned} \tag{5.32}$$

Perhitungan 8-Level *Forward Recursion Metric* dan *Posteriori* LLR

Untuk $z = 0, 1, \ldots, \frac{K}{8} - 1$ menggunakan persamaan sebagai berikut.

$$\begin{aligned}
A_{(8\times z)+1}(S_i) &= max\{\big(A_{(8\times z)}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(A_{(8\times z)}(S_{j1}) + \Gamma_{ij1}\big)\} \\
A_{(8\times z)+2}(S_i) &= max\{\big(A_{(8\times z)+1}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(A_{(8\times z)+1}(S_{j1}) + \Gamma_{ij1}\big)\} \\
A_{(8\times z)+3}(S_i) &= max\{\big(A_{(8\times z)+2}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(A_{(8\times z)+2}(S_{j1}) + \Gamma_{ij1}\big)\} \\
A_{(8\times z)+4}(S_i) &= max\{\big(A_{(8\times z)+3}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(A_{(8\times z)+3}(S_{j1}) + \Gamma_{ij1}\big)\} \\
A_{(8\times z)+5}(S_i) &= max\{\big(A_{(8\times z)+4}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(A_{(8\times z)+4}(S_{j1}) + \Gamma_{ij1}\big)\} \\
A_{(8\times z)+6}(S_i) &= max\{\big(A_{(8\times z)+5}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(A_{(8\times z)+5}(S_{j1}) + \Gamma_{ij1}\big)\} \\
A_{(8\times z)+7}(S_i) &= max\{\big(A_{(8\times z)+6}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(A_{(8\times z)+6}(S_{j1}) + \Gamma_{ij1}\big)\} \\
A_{(8\times z)+8}(S_i) &= max\{\big(A_{(8\times z)+7}(S_{j0}) + \Gamma_{ij0}\big) \quad , \big(A_{(8\times z)+7}(S_{j1}) + \Gamma_{ij1}\big)\}
\end{aligned} \tag{5.33}$$

, dengan nilai inisialisasi $A_0(S_0) = 0$ dan $A_0(S_i) = -\infty, i > 0$.

Untuk $t = 0, 1, \ldots, \frac{K}{8} - 1$ dan menggunakan persamaan 5.32 dan 5.33, maka dida-

patkan persamaan *posteriori* LLR yang dioptimasi menjadi sebagai berikut.

$$
\begin{aligned}
L(u_{(8\times t)}|y) \approx & \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)}=+1}(A_{(8\times t)}(\acute{s}) + B_{(8\times t)+1}(s) + \Gamma_{(8\times t)+1}(\acute{s},s)) - \\
& \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)}=-1}(A_{(8\times t)}(\acute{s}) + B_{(8\times t)+1}(s) + \Gamma_{(8\times t)+1}(\acute{s},s)) \\
L(u_{(8\times t)+1}|y) \approx & \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+1}=+1}(A_{(8\times t)+1}(\acute{s}) + B_{(8\times t)+2}(s) + \Gamma_{(8\times t)+2}(\acute{s},s)) - \\
& \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+1}=-1}(A_{(8\times t)+1}(\acute{s}) + B_{(8\times t)+2}(s) + \Gamma_{(8\times t)+2}(\acute{s},s)) \\
L(u_{(8\times t)+2}|y) \approx & \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+2}=+1}(A_{(8\times t)+2}(\acute{s}) + B_{(8\times t)+3}(s) + \Gamma_{(8\times t)+3}(\acute{s},s)) - \\
& \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+2}=-1}(A_{(8\times t)+2}(\acute{s}) + B_{(8\times t)+3}(s) + \Gamma_{(8\times t)+3}(\acute{s},s)) \\
L(u_{(8\times t)+3}|y) \approx & \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+3}=+1}(A_{(8\times t)+3}(\acute{s}) + B_{(8\times t)+4}(s) + \Gamma_{(8\times t)+4}(\acute{s},s)) - \\
& \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+3}=-1}(A_{(8\times t)+3}(\acute{s}) + B_{(8\times t)+4}(s) + \Gamma_{(8\times t)+4}(\acute{s},s)) \\
L(u_{(8\times t)+4}|y) \approx & \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+4}=+1}(A_{(8\times t)+4}(\acute{s}) + B_{(8\times t)+5}(s) + \Gamma_{(8\times t)+5}(\acute{s},s)) - \\
& \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+4}=-1}(A_{(8\times t)+4}(\acute{s}) + B_{(8\times t)+5}(s) + \Gamma_{(8\times t)+5}(\acute{s},s)) \\
L(u_{(8\times t)+5}|y) \approx & \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+5}=+1}(A_{(8\times t)+5}(\acute{s}) + B_{(8\times t)+6}(s) + \Gamma_{(8\times t)+6}(\acute{s},s)) - \\
& \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+5}=-1}(A_{(8\times t)+5}(\acute{s}) + B_{(8\times t)+6}(s) + \Gamma_{(8\times t)+6}(\acute{s},s)) \\
L(u_{(8\times t)+6}|y) \approx & \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+6}=+1}(A_{(8\times t)+6}(\acute{s}) + B_{(8\times t)+7}(s) + \Gamma_{(8\times t)+7}(\acute{s},s)) - \\
& \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+6}=-1}(A_{(8\times t)+6}(\acute{s}) + B_{(8\times t)+7}(s) + \Gamma_{(8\times t)+7}(\acute{s},s)) \\
L(u_{(8\times t)+7}|y) \approx & \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+7}=+1}(A_{(8\times t)+7}(\acute{s}) + B_{(8\times t)+8}(s) + \Gamma_{(8\times t)+8}(\acute{s},s)) - \\
& \max_{(\acute{\mathbf{s}},\mathbf{s})\Rightarrow u_{(8\times t)+7}=-1}(A_{(8\times t)+7}(\acute{s}) + B_{(8\times t)+8}(s) + \Gamma_{(8\times t)+8}(\acute{s},s))
\end{aligned}
\tag{5.34}
$$

5.6.3. Top Level Arsitektur Turbo Dekoder

Top level arsitektur yang ditunjukkan pada gambar 5.4 dioptimasi dengan cara menggunakan ulang komponen dekoder yang telah dioptimasi dan menggunakan *dual bank ram* untuk blok *interleaver*. Gambar 5.10 menunjukkan top level arsitekturnya. Dengan blok `PROPOSED DECODER` merupakan implementasi dari persamaan 5.34 dengan illustrasi arsitekturnya ditunjukkan pada gambar 5.13.

Gambar 5.12: Top level arsitektur turbo dekoder yang dioptimasi

Gambar 5.13: Blok arsitektur perhitungan *posteriori* LLR yang telah dioptimasi (persamaan 5.34) atau bagian dari blok `PROPOSED DECODER` pada gambar 5.12

BAB 6

PERANCANGAN MODEL FUNGSIONAL KODE TURBO

Pada bab sebelumnya telah dijelaskan mengenai teori kode turbo, pemodelan sistem dengan menggunakan systemC, ABV, dan optimasi perancangan HW kode turbo. Pada bab ini dan bab selanjutnya akan dijelaskan mengenai implementasi dari yang telah dijelaskan pada bab sebelumnya. Pada bab 6 ini akan dijelaskan mengenai perancangan model fungsional kode turbo. Karena didapatkan kode yang harus dioptimasi dalam bahasa MATLAB baik enkoder maupun dekoder, maka tahapan perancangan model fungsional yang telah dioptimasi diimplementasi dalam bahasa MATLAB. Selain itu, penggunaan bahasa ini digunakan karena kemudahan dalam mengukur performa BER-nya. Kemudian dari model ini dilakukan proses translasi ke systemC.

6.1. Model Fungsional Kode Turbo MATLAB

Pada bagian ini akan dirancang model fungsional kode turbo yang telah dioptimasi menggunakan persamaan-persamaan yang telah didapatkan pada bab 5.

6.1.1. Model Fungsional Turbo Enkoder MATLAB

Tahap perancangan model fungsional untuk turbo enkoder ini tidak terlalu rumit karena hanya melakukan translasi persamaan 5.25 dan 5.27 ke dalam bahasa MATLAB. Contoh program di bawah adalah potongan kode yang hasil translasi tersebut.

Program 6.1: Potongan Program Model Fungsional Turbo Enkoder pada MATLAB

```
for i = 0:(( length(Ut)/8)-1)

    S1 = Xt(1) + Xt(2);
    S2 = Xt(3) + Ut((8*i)+1);
    S3 = Ut((8*i)+2) + Ut((8*i)+3);
    S4 = Xt(1) + Ut((8*i)+3);
    S5 = Ut((8*i)+4) + Ut((8*i)+5);

    P1 = Xt(2) + S2;
```

```
    P2 = Xt(2) + S3;
    P3 = Ut((8*i)+2) + S1;
    P4 = Ut((8*i)+6) + S5;

    Q1 = S2 + S3;
    Q2 = P1 + P4;
...
```

Pada program 6.1 ditunjukkan pula panjang iterasi pada perhitungan komponen enkoder memiliki tingkat paralelisasi 8-level.

Sedangkan *platform* verifikasi yang digunakan untuk model fungsional ini ditunjukkan oleh gambar 6.1. Skenario verifikasi pada model ini adalah memvariasikan untuk semua nilai blok K pada input random seperti yang ditunjukkan pada tabel 2.1 dan mengulanginya hingga 10 kali untuk meyakinkan bahwa model yang telah dirancang identik dengan model acuan. Parameter identik didapat jika hasil *counter* benar bernilai $188 \times 10 \;=\; 1880$.

Program di bawah merupakan potongan dari *platform* verifikasi untuk model fungsional turbo enkoder ini.

Program 6.2: Potongan Program *Platform* Model Fungsional Turbo Enkoder pada MATLAB

```
for j = 0 : 9
disp          ('V//////////////////////////\\\\\\\\\\\\\\\\\\\\V')
disp ([' LEVEL OF CONFIDENCE : ' num2str(10*(j+1)) '%'])
        for i = 1:length(K)

                l = K(i);

                % DATA MASUKAN BERUPA BILANGAN RANDOM 1 BIT
                input = randint(1,l);

                % TURBO ENCODER YANG TELAH DIOPTIMASI
                [output1a,output2a,output3a]=turbo_enc_m(input);
                data1 = [output1a,output2a,output3a];

                % TURBO ENCODER ACUAN
                [output1b,output2b,output3b]=turbo_enc(input) ;
                data2 = [output1b,output2b,output3b];

                % VERIFIKASI
                if ( isequal(data1,data2) )
                        benar = benar+1;
                else
                        salah = salah+1;
                end
```

```
                    disp         ('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')
                    disp ([' LEVEL OF CONFIDENCE :' num2str(10*(j+1)) '%'])
                    disp ([' K :  '  num2str(l )])
                    disp ([' benar  :  '  num2str(benar )])
                    disp ([' salah  :  '  num2str( salah )])
                    disp ('           iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii            ')

          end
disp           ('\\\\\\\\\\\\\\\\\\\\\\\\/////////////////////')

end
 ...
```

Gambar 6.1: *Platform* verifikasi model fungsional tubo enkoder MATLAB

Berikut adalah hasil potongan eksekusi program di atas. Hasil yang didapat untuk semua kemungkinan yang didapat model yang telah dioptimasi identik dengan model acuan. Gambar 6.2 menunjukkan hasil *print screen* pada *window* MATLAB.

```
\////////////////////\\\\\\\\\\\\\\\\/
LEVEL OF CONFIDENCE : 10%
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
LEVEL OF CONFIDENCE : 10%
K : 40
benar : 1
salah : 0
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
LEVEL OF CONFIDENCE : 10%
K : 48
benar : 2
salah : 0
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
...
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
LEVEL OF CONFIDENCE : 100%
K : 6144
benar : 1880
salah : 0
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
\\\\\\\\\\\\\\\\\\\\\\/////////////////
```

Gambar 6.2: *Print Screen* verifikasi model fungsional tubo enkoder MATLAB

6.1.2. Model Fungsional Turbo Dekoder MATLAB

Tahap perancangan algoritma Max-log MAP dan SWA dilakukan dengan memecahnya menjadi modul-modul kecil, yaitu modul untuk menghitung *backward recursion metric* yang mengacu pada persamaan 5.32, modul untuk menghitung *forward recursion metric* dan *posteriori* LLR sekaligus yang mengacu pada persamaan 5.33 dan 5.14, dan top modul untuk memodelkan kedua algoritma tersebut dengan cara memanggil submodul dan memecah input untuk submodul tersebut.

Program 6.3 merupakan potongan program submodul untuk menghitung *backward recursion metric*. Submodul tersebut memiliki parameter masukan batas kanan perhitungan *backward recursion metric* dan batas kirinya yang dinotasikan dengan `kanan` dan `kiri`. Selain itu variabel `code1` menotasikan $V_1(X_k)$ atau $V_2(X'_k)$, sedangkan `code2` menotasikan $R(Z_k)$ atau $R(Z'_k)$ seperti yang ditunjukkan pada gambar 5.12.

Program 6.3: Potongan Program Perhitungan *Backward Recursion Metric* pada MATLAB

```
function [beta] = betacomp(kanan, kiri ,code1,code2)

ITERASI = (kanan-kiri+1)/8;
offset = kiri -1;

for k=ITERASI:-1:1
...

branch1=beta(1,8*k+offset-1+1);
branch2=beta(5,8*k+offset-1+1)+code2(8*k+offset-1)+code1(8*k+offset-1);
beta_buf(1,8*k+offset-1)=max(branch1,branch2);

branch1=beta(1,8*k+offset-1+1)+code2(8*k+offset-1)+code1(8*k+offset-1);
branch2=beta(5,8*k+offset-1+1);
beta_buf(2,8*k+offset-1)=max(branch1,branch2);

branch1=beta(2,8*k+offset-1+1)+code1(8*k+offset-1);
branch2=beta(6,8*k+offset-1+1)+code2(8*k+offset-1);
beta_buf(3,8*k+offset-1)=max(branch1,branch2);

branch1=beta(2,8*k+offset-1+1)+code2(8*k+offset-1);
branch2=beta(6,8*k+offset-1+1)+code1(8*k+offset-1);
beta_buf(4,8*k+offset-1)=max(branch1,branch2);

branch1=beta(3,8*k+offset-1+1)+code2(8*k+offset-1);
branch2=beta(7,8*k+offset-1+1)+code1(8*k+offset-1);
beta_buf(5,8*k+offset-1)=max(branch1,branch2);

branch1=beta(3,8*k+offset-1+1)+code1(8*k+offset-1);
branch2=beta(7,8*k+offset-1+1)+code2(8*k+offset-1);
beta_buf(6,8*k+offset-1)=max(branch1,branch2);
```

```
branch1=beta(4,8*k+offset-1+1)+code2(8*k+offset-1)+code1(8*k+offset-1);
branch2=beta(8,8*k+offset-1+1);
beta_buf(7,8*k+offset-1)=max(branch1,branch2);

branch1=beta(4,8*k+offset-1+1);
branch2=beta(8,8*k+offset-1+1)+code2(8*k+offset-1)+code1(8*k+offset-1);
beta_buf(8,8*k+offset-1)=max(branch1,branch2);

beta=beta_buf;
...
```

Sedangkan, program 6.4 merupakan potongan program submodul untuk menghitung *forward recursion metric* dan *posteriori* LLR sekaligus. Submodul tersebut memiliki parameter masukan batas kiri perhitungan *forward recursion metric* dan batas kanannya yang dinotasikan dengan lambang `kiri` dan `kanan`. Selain itu variabel `code1` dan `code2` menotasikan hal yang sama seperti pada submodul sebelumnya. Pada submodul ini juga dibutuhkan informasi *backward recursion metric* yang telah dihitung sebelumnya dan informasi *forward recursion metric* subblok sebelumnya yang dinotasikan dengan variabel `in`.

Program 6.4: Potongan Program Perhitungan *Forward Recursion Metric* dan *posteriori* LLR pada MATLAB

```
function [L temp] = alphacomp(kiri,kanan,code1,code2,beta,in)

ITERASI = ((kanan-kiri+1)/8)-1;
offset = kiri-1;

for k=0:ITERASI
...

branch1=alpha(1,8*k+offset+3);
branch2=alpha(2,8*k+offset+3)+code2(8*k+offset+3)+code1(8*k+offset+3);
alpha_buf(1,8*k+offset+3+1)=max(branch1,branch2);

branch1=alpha(4,8*k+offset+3)+code2(8*k+offset+3);
branch2=alpha(3,8*k+offset+3)+code1(8*k+offset+3);
alpha_buf(2,8*k+offset+3+1)=max(branch1,branch2);

branch1=alpha(5,8*k+offset+3)+code2(8*k+offset+3);
branch2=alpha(6,8*k+offset+3)+code1(8*k+offset+3);
alpha_buf(3,8*k+offset+3+1)=max(branch1,branch2);

branch1=alpha(8,8*k+offset+3);
branch2=alpha(7,8*k+offset+3)+code2(8*k+offset+3)+code1(8*k+offset+3);
alpha_buf(4,8*k+offset+3+1)=max(branch1,branch2);

branch1=alpha(2,8*k+offset+3);
branch2=alpha(1,8*k+offset+3)+code2(8*k+offset+3)+code1(8*k+offset+3);
```

```
alpha_buf(5,8*k+offset+3+1)=max(branch1,branch2);

branch1=alpha(3,8*k+offset+3)+code2(8*k+offset+3);
branch2=alpha(4,8*k+offset+3)+code1(8*k+offset+3);
alpha_buf(6,8*k+offset+3+1)=max(branch1,branch2);

branch1=alpha(6,8*k+offset+3)+code2(8*k+offset+3);
branch2=alpha(5,8*k+offset+3)+code1(8*k+offset+3);
alpha_buf(7,8*k+offset+3+1)=max(branch1,branch2);

branch1=alpha(7,8*k+offset+3);
branch2=alpha(8,8*k+offset+3)+code2(8*k+offset+3)+code1(8*k+offset+3);
alpha_buf(8,8*k+offset+3+1)=max(branch1,branch2);

alpha=alpha_buf;

plus(1) =beta(1,8*k+offset+4+1)+code2(8*k+offset+4)+code1(8*k+offset+4)+alpha(2,8*k+offset+4);
plus(2) =beta(2,8*k+offset+4+1)+code1(8*k+offset+4)+alpha(3,8*k+offset+4);
plus(3) =beta(3,8*k+offset+4+1)+code1(8*k+offset+4)+alpha(6,8*k+offset+4);
plus(4) =beta(4,8*k+offset+4+1)+code2(8*k+offset+4)+code1(8*k+offset+4)+alpha(7,8*k+offset+4);
plus(5) =beta(5,8*k+offset+4+1)+code2(8*k+offset+4)+code1(8*k+offset+4)+alpha(1,8*k+offset+4);
plus(6) =beta(6,8*k+offset+4+1)+code1(8*k+offset+4)+alpha(4,8*k+offset+4);
plus(7) =beta(7,8*k+offset+4+1)+code1(8*k+offset+4)+alpha(5,8*k+offset+4);
plus(8) =beta(8,8*k+offset+4+1)+code2(8*k+offset+4)+code1(8*k+offset+4)+alpha(8,8*k+offset+4);


minus(1)=beta(1,8*k+offset+4+1)+alpha(1,8*k+offset+4);
minus(2)=beta(2,8*k+offset+4+1)+code2(8*k+offset+4)+alpha(4,8*k+offset+4);
minus(3)=beta(3,8*k+offset+4+1)+code2(8*k+offset+4)+alpha(5,8*k+offset+4);
minus(4)=beta(4,8*k+offset+4+1)+alpha(8,8*k+offset+4);
minus(5)=beta(5,8*k+offset+4+1)+alpha(2,8*k+offset+4);
minus(6)=beta(6,8*k+offset+4+1)+code2(8*k+offset+4)+alpha(3,8*k+offset+4);
minus(7)=beta(7,8*k+offset+4+1)+code2(8*k+offset+4)+alpha(6,8*k+offset+4);
minus(8)=beta(8,8*k+offset+4+1)+alpha(7,8*k+offset+4);

L(8*k+offset+4)= max(plus)-max(minus);
 ...
temp=alpha_buf(1:8,kanan+1);
 ...
```

Top modul seperti yang ditunjukkan pada program 6.5 memanggil kedua submodul di atas. Nilai `WINDOW_LENGTH` $= 40$ menunjukkan panjang dari subblok SWA, sedangkan nilai 8 menunjukkan panjang *guard window*. Variabel `L` menunjukkan *posteriori* LLR yang telah dihitung.

Program 6.5: Potongan Program Implementasi Algoritma Max-log-MAP dan SWA pada MATLAB

```
l=length(code1);
package=l;

l = floor (package/WINDOW_LENGTH);
```

```
for i = 0:(l-1)
i_temp = WINDOW_LENGTH*i;
 ...

[beta (:,1: i_temp+WINDOW_LENGTH+8)] = betacomp(i_temp+WINDOW_LENGTH+8,i_temp+1,code1,code2);
[temp(1:i_temp+WINDOW_LENGTH) alpha(:,i_temp+WINDOW_LENGTH+1)] = alphacomp(i_temp+1,i_temp+...
        WINDOW_LENGTH,code1,code2,beta,alpha(:,i_temp+1));
L(i_temp+1:i_temp+WINDOW_LENGTH)=temp(i_temp+1:i_temp+WINDOW_LENGTH);
 ...
```

Metode verifikasi untuk model fungsional turbo dekoder ini berbeda dengan apa metode verifikasi yang dilakukan pada turbo enkoder. Pada turbo dekoder ini perbandingannya dengan melihat kurva performa BER. Tujuan dari tugas akhir ini untuk modul turbo dekoder adalah salah satunya memiliki performa BER yang lebih baik. Parameter lebih baiknya untuk performa BER ini adalah untuk besar *Signal-to-Noise Ratio* (SNR) atau $\frac{Eb}{No}$ yang sama nilai BERnya lebih kecil atau memiliki $\frac{Eb}{No}$ yang lebih kecil untuk nilai BER yang sama.

Berbeda dengan turbo enkoder, model fungsional ini hanya memiliki dua skenario verifikasi, yaitu untuk $K = 200$ dengan banyaknya sample sebangak 1000 dicari $\frac{Eb}{No}$ hingga sistem memiliki performa BER 10^{-5} dan untuk $K = 40$ dengan banyaknya sample sebangak 10000 dicari $\frac{Eb}{No}$ hingga sistem memiliki performa BER 10^{-5}. Selain itu pada bahasan ini akan ditampilkan juga hasil simulasi untuk panjang *guard window* $= 0$.

Seperti yang telah dijelaskan pada bab 1 bahwa model kanal yang digunakan hanya terbatas pada modulasi BPSK dengan model derau AWGN. Gambar 6.3 merangkum *platform* verifikasi yang digunakan. Serta program 6.6 menunjukkan potongan *platform* verifikasinya.

Gambar 6.3: *Platform* verifikasi model fungsional tubo dekoder MATLAB

Program 6.6: Potongan Program *Platform* Model Fungsional Turbo Dekoder pada MATLAB

```
len    = 200;
sample = 1000;

for i=1:sample;
datain = randint (1, len );
[w,e,r]=turbo_enc( datain );
```

```
w_BPSK = 2*w-1;
e_BPSK = 2*e-1;
r_BPSK = 2*r-1;

llr 2 = awgn(w_BPSK,dB_value,'measured','dB');
llr 3 = awgn(e_BPSK,dB_value,'measured','dB');
llr 4 = awgn(r_BPSK,dB_value,'measured','dB');

variance = 1/(2*(10^(dB_value/10)));
llr 2 = 2*llr 2/variance;
llr 3 = 2*llr 3/variance;
llr 4 = 2*llr 4/variance;
...

[Lm40 max_beta_temp(i) min_beta_temp(i)] = turbo_dec_m(llr 2, llr 3, llr 4, iteration _value, WINDOW_LENGTH40);
Lm40 = double(Lm40 > 0);
error 40= biterr (Lm40,datain);
c_summ40=c_summ40+error40;
...
```

Program di atas menunjukkan potongan program yang menjelaskan dari proses *encoding*, implementasi model kanal yang digunakan, proses *decoding*, dan pengukuran performa BER. Gambar 6.4, 6.5, dan 6.6 menunjukkan hasil dari skenario verifikasi yang telah dijelaskan sebelumnya. Garis biru menunjukkan turbo dekoder yang dioptimasi. Dari yang didapat pada gambar 6.4 dan 6.5 turbo dekoder memiliki hasil performa BER yang lebih baik. Namun gambar 6.6 menunjukkan hasil yang tidak lebih baik karena panjang *guard window*-nya $= 0$.

Gambar 6.4: Kurva BER model fungsional tubo dekoder $K = 200$ MATLAB

Gambar 6.5: Kurva BER model fungsional tubo dekoder $K = 40$ MATLAB

Gambar 6.6: Kurva BER model fungsional tubo dekoder $K = 200$ dengan *guard window* $= 0$ MATLAB

6.2. Model Fungsional Kode Turbo SystemC

Setelah pada pemodelan sebelumnya diyakini bahwa kode turbo yang dioptimasi memiliki performa yang lebih baik, maka pemodelan selanjutnya adalah pemodelan dengan tingkat akurasi waktu dan port yang tinggi dengan menggunakan systemC sebelum ke perancangan HW menggunakan HDL.

6.2.1. Model Fungsional Turbo Enkoder SystemC

Gambar 6.7 merupakan pecahan submodul dengan tingkat akurasi waktu dan port yang tinggi (model RTL) dari gambar 6.7. Terdapat 3 blok *datapath* utama yang membangun keseluruhan sistem turbo enkoder yang telah dioptimasi sebelumnya, yaitu *dual bank ram*, *index generator interleaver*, dan *proposed constituent encoder*. Ketiga modul ini mendukung pengolahan data 8-level paralelisasi.

Gambar 6.7: Submodul gambar 5.10 untuk model RTL systemC

Seperti yang telah dijelaskan sebelumnya, untuk meningkatkan kecepatan turbo enkoder hingga mencapai maksimal 16 kali lebih cepat maka digunakan blok *dual bank ram*. Oleh karena itu ditunjukkan pada gambar 6.7 bahwa blok *dual bank ram* memiliki port `bank_w` atau `bank_r` untuk mengatur di `mem0` atau `mem1` data akan disimpan, sehingga tidak terjadi bentrokan data antar urutan data masuk.

Sedangkan untuk kedua blok lainnya mengikuti persamaan 5.25 dan 5.27. Untuk mendukung tingkat akurasi waktunya kedua blok ini menggunakan gabungan blok kombinasional dan *register*. Selain itu dibutuhkan blok FSM sebagai unit kendali semua blok *datapath*.

Blok FSM ditunjukkan seperti pada gambar 6.8. Berikut adalah deskripsi untuk masing-masing *state*-nya.

- MULAI :
  State ini merupakan *state* inisialisasi untuk semua *register* sehingga nilainya bernilai 0. *Trigger* utama untuk *state* ini untuk pindah ke *state* berikutnya PREOLAH_INIT adalah sinyal `start` dan `K_valid` yang menandakan bahwa proses *encoding* dimulai dan nilai K yang diterima valid.
- PREOLAH_INIT :
  Pada *state* ini sistem dalam kondisi siap menerima input hingga `in_valid` bernilai 1 yang berarti *state* pindah ke *state* berikutnya yaitu OLAH_INIT.
- OLAH_INIT :
  Pada *state* ini sistem melakukan proses penyimpanan untuk urutan data yang pertama. Proses ini terus berlangsung selama inputnya valid atau semua input telah tersimpan yaitu sebanyak K bit. Namun jika belum semua input tersimpan dan nilai inputnya tidak valid, maka *state* ini akan kembali ke *state* sebelumnya

yakni PREOLAH_INIT hingga inputnya kembali valid. Setelah semua input tersimpan, maka *state* selanjutnya adalah COMPARE.

- COMPARE :
  Setelah urutan data pertama tersimpan, maka *state* ini memeriksa apakah urutan data kedua memiliki K yang sama dengan urutan data pertama. Jika sama, maka *state* selanjutnya mengolah kedua urutan ini dalam jumlah *clock* yang sama (proses *encoding* untuk urutan data pertama dan menyimpan urutan data kedua). Jika tidak sama, maka *state* selanjutnya menyimpan urutan data kedua dan melakukan proses *encoding* urutan data pertama, kemudian melakukan proses *encoding* untuk urutan data kedua ini.

- PREOLAH_NEXT_SEQ dan OLAH_NEXT_SEQ:
  State PREOLAH_NEXT_SEQ dan OLAH_NEXT_SEQ memiliki proses yang hampir mirip dengan *state* PREOLAH_INIT dan OLAH_INIT, namun perbedaannya adalah selain menyimpan urutan data berikutnya yang akan diproses, *state* ini juga melakukan proses *encoding* urutan data yang telah disimpan sebelumnya.

- PREOLAH_PREFINISHING dan OLAH_PREFINISHING:
  State PREOLAH_PREFINISHING dan OLAH_PREFINISHING memiliki proses yang hampir mirip dengan *state* PREOLAH_NEXT_SEQ dan mirip juga dengan *state* OLAH_NEXT_SEQ, namun perbedaannya adalah total *clock* yang berbeda karena*state* ini terjadi saat ukuran K urutan data berikutnya tidak sama dengan ukuran K urutan data sebelumnya.

- PREOLAH_FINISHING dan OLAH_FINISHING:
  Kedua *state* ini melakukan proses *encoding* untuk urutan data yang terakhir yang telah disimpan sebelumnya. PREOLAH_FINISHING melakukan proses *idle* yang artinya semua blok *datapath* tidak melakukan sesuatu apapun. Proses ini diperlukan pada *state* OLAH_FINISHING yang melakukan proses *encoding* terakhir .

Program 6.7 merupakan potongan program top level arsitektur untuk model fungsional systemC yang merupakan gabungan antara *datapath* dan FSM yang telah dijelaskan di atas. Port $output\_ext1, \ldots, output\_ext4$ menunjukkan implementasi proses *multiplexing* seperti yang ditunjukkan pada persamaan 2.6.

Program 6.7: Potongan Program Top Level Arsitektur Model Fungsional Turbo Enkoder pada SytemC

```
SC_MODULE (turbo_encoder) {
 ...
FSM_enc          fsm;
datapath_enc     datapath ;
 ...
SC_CTOR(turbo_encoder) : fsm("FSM"),datapath("DATAPATH")
  {

    fsm.clock              (clock            );
    fsm. reset _in         ( reset _in       );
    fsm. start             ( start           );
    fsm.K_valid            (K_valid          );
    fsm.in_valid           (in_valid         );
    fsm.K_in               (K_in             );
    fsm. reset             ( reset           );
    fsm.cs                 (cs               );
    fsm.we                 (we               );
    fsm.oe                 (oe               );
    fsm.bank_w             (bank_w           );
    fsm.bank_r             (bank_r           );
    fsm. int _en           ( int _en         );
    fsm.count_en           (count_en         );
    fsm.conv_en            (conv_en          );
    fsm. tail _valid       ( tail _valid     );
    fsm.out_valid          (out_valid        );
    fsm.in_out_valid       (in_out_valid     );
    fsm.K                  (K                );

    datapath . clock       (clock            );
    datapath . reset       ( reset           );
    datapath . int _en     ( int _en         );
    datapath . count_en    (count_en         );
    datapath . conv_en     (conv_en          );
    datapath . K           (K                );
    datapath . cs          (cs               );
    datapath . we          (we               );
    datapath . oe          (oe               );
    datapath . bank_w      (bank_w           );
    datapath . bank_r      (bank_r           );
    datapath . data_in0    (data_in0         );
    datapath . data_in1    (data_in1         );
    datapath . data_in2    (data_in2         );
    datapath . data_in3    (data_in3         );
    datapath . data_in4    (data_in4         );
    datapath . data_in5    (data_in5         );
    datapath . data_in6    (data_in6         );
    datapath . data_in7    (data_in7         );
    datapath . output_0    (output_0         );
    datapath . output_1    (output_1         );
    datapath . output_2    (output_2         );
    datapath . output_3    (output_3         );
```

```
    datapath.output_4      (output_4      );
    datapath.output_5      (output_5      );
    datapath.output_6      (output_6      );
    datapath.output_7      (output_7      );
    datapath.output_ext1   (output_ext1   );
    datapath.output_ext2   (output_ext2   );
    datapath.output_ext3   (output_ext3   );
    datapath.output_ext4   (output_ext4   );

...
```

Gambar 6.8: FSM model fungsional turbo enkoder systemC

Seperti yang telah dijelaskan pada bab 4 bahwa untuk model fungsional dengan menggunakan systemC akan menggunakan metode verifikasi *average case* dan *corner case*. Titik *average case* dipilih dengan nilai $K = 232$ (urutan data pertama K_1 dan kedua K_2) dan $K = 200$ (urutan data ketiga K_3) dengan skenario pertama adalah urutan data pertama dan kedua menjadi dua urutan data pertama yang masuk dan diakhiri dengan urutan data ketiga. Sedangkan skenario kedua adalah urutan data ketiga masuk duluan, kemudian diakhiri urutan data kedua. Skenario pertama mewakili skenario untuk perpindahan panjang urutan data dari yang lebih besar ke yang lebih kecil. Namun sebaliknya untuk skenario kedua. Sedangkan titik *corner case* dipilih urutan data dengan nilai $K = 6144$ (urutan data pertama dan kedua) dan $K = 40$ (urutan data ketiga) dimana merupakan panjang urutan data terbesar dan terkecil (lihat table 2.1).

Berbeda dengan metode verifikasi untuk model fungsional turbo enkoder yang awal yaitu dalam bahasa MATLAB. Turbo enkoder acuan untuk metode verifikasi sekarang adalah model MATLAB yang telah dibuat tersebut, yaitu model turbo enkoder yang telah dioptimasi. Metode perbandingannya dengan membandingkan hasil yang didapat di MATLAB dalam bentuk file *dump* yang dibandingkan dengan yang didapat model fungsional dengan model RTL pada systemC dan dalam *environment* systemC. Gambar *platform* verifikasinya ditunjukkan seperti gambar 6.9.

Gambar 6.9: *Platform* verifikasi model fungsional tubo enkoder systemC

Model fungsional systemC ini dapat disimpulkan benar jika hasil perhitungan variabel $benar$ bernilai $K_1/8 + K_2/8 + K_3/8$ (skenario pertama) ditambah $K_3/8 + K_2/8$ (skenario kedua) dengan nilai $salah = 0$. Hasil verifikasi untuk titik *average case* dapat dilihat pada gambar 6.10 yang menunjukkan nilai $benar = 137 = 232/8 + 232/8 + 200/8 + 200/8 + 232/8$ dengan nilai $salah = 0$. Sedangkan untuk titik *corner case*

ditunjukkan pada gambar 6.11 dengan nilai $benar = 2314 = 6144/8 + 6144/8 + 40/8 + 40/8 + 6144/8$ dengan nilai $salah = 0$. Kedua hasil tersebut mengindikasikan bahwa model fungsional tersebut benar dan siap untuk diimplementasikan dalam HDL. Selain itu pada kedua gambar tersebut dapat dilihat pula awan biru dan ungu yang menunjukkan skenario keberapa yang sedang berjalan dan panjang urutan data K yang sedang diproses.

6.2.2. Model Fungsional Turbo Dekoder SystemC

Dalam proses perancangan model fungsional turbo dekoder dalam systemC perlu diperhatikan lebih detil mengenai tipe data karena data yang diolah merupakan data *soft decision* dan data masukan untuk blok turbo dekoder merupakan keluaran dari model saluran (modulasi BPSK dan model derau AWGN). Pada dasarnya penambahan model derau AWGN menjadikan sinyal keluaran turbo enkoder dan modulasi BPSK yang deterministik menjadi nondeterministik (random). Oleh karena itu, pada perancangan tugas akhir ini kita menggunakan format data *fixed point*. Namun, data random memiliki data maksimal tak hingga dan data minimalnya minus tak hingga. Sehingga, pada tugas akhir ini digunakan metode sampling data percobaan model saluran dengan nilai SNR terbatas dari 0 dB hingga 4.8 dB. Pemilihan batas nilai SNR ini karena melihat untuk performa BER yang paling jelek yang didapat untuk $K = 40$ (lihat gambar 6.5) hanya terbatas pada jangkauan SNR 0 dB hingga 4.8 dB.

Metode sampling yang dilakukan diulangi hingga 1000 kali pengambilan data seperti yang ditunjukkan pada program 6.8. Nilai yang didapat tidak pernah lebih dari nilai 50 dan tidak pernah kurang dari -50. Oleh karena itu dipilih panjang bit integernya adalah 7 bit.

Program 6.8: Program Sampling Nilai Maksimum dan Minimum Keluaran Model Saluran BPSK dan AWGN pada MATLAB

```
clear all
clc

i = 1;

for dB_value = 0:0.4:4.8
    for j = 1:1000
        datain = randint(1, 6144);

        w_BPSK = 2*datain-1;

        %AWGN using MATLAB function
```

```
            llr 2 = awgn(w_BPSK,dB_value,'measured','dB');

            variance = 1/(2*(10^(dB_value/10)));
            llr 2 = 2* llr 2/ variance ;

           temp(j ,:) = llr 2;

      end

      maks(i) = max(max(temp));
      mini(i) = min(min(temp));
      i=i+1;
end

max(maks)
min(mini)

 ...
```

Gambar 6.10: Hasil simulasi titik *average case* ($K_1 = K_2 = 232$ dan $K_3 = 200$) model fungsional turbo enkoder systemC

Gambar 6.11: Hasil simulasi titik *corner case* ($K_1 = K_2 = 6144$ dan $K_3 = 40$) model fungsional turbo enkoder systemC

Penentuan nilai maksimal dan minimal ini juga dilakukan untuk variabel *backward* dan *forward recursion metric* serta *posteriori* LLR. Namun metode yang digunakan berbeda dengan program 6.8. Ketiga variabel memiliki nilai maksimal ketika ukuran blok $K = 6144$ (K maksimum tabel 2.1) dengan input turbo enkoder memiliki nilai 1 semua. Sedangkan, untuk nilai minimumnya didapat untuk ukuran blok $K = 6144$ dengan nilai input turbo enkoder memiliki nilai 0 semua. Hasil penentuan nilai maksimum dan minimumnya dapat dilihat di tabel 6.1.

Tabel 6.1: Tabel nilai maksimum dan maksimum $B_k(s)$, $A_{k-1}(\acute{s})$, dan $L(u_k|y)$

Variabel	Nilai Maksimum	Nilai Minimum	Bit Integer
$B_k(s)$	$3.884091276810001e+03$	$-1.225959853966581e+02$	$13\ bit$
$A_{k-1}(\acute{s})$	$3.599512921592176e+05$	$-2.202063114077681e+02$	$17\ bit$
$L(u_k\|y)$	$1.861585674031521e+02$	$-1.968187536879815e+02$	$9\ bit$

Sedangkan untuk penentuan bit *fraction*-nya perlu dilihat pengaruhnya terhadap performa BER turbo dekoder. Karena pada proses HDD hanya dilihat tanda dari nilai *posteriori* LLR, maka tidak diperlukan tingkat presisi yang tinggi. Oleh karena itu,

pada tugas akhir ini tidak digunakan bit *fraction*. Hasil performa BERnya dapat dilihat pada gambar 6.12 dan 6.13. Jika dibandingkan dengan model fungsional tanpa *fixed point* yaitu gambar 6.4 dan 6.5, hasil yang didapat dengan format data *fixed point* memiliki performa yang lebih jelek, namun masih memiliki performa yang lebih baik dibanding model acuan.

Gambar 6.12: Kurva BER model fungsional tubo dekoder $K = 200$ tanpa bit *fraction* systemC

Gambar 6.13: Kurva BER model fungsional tubo dekoder $K = 40$ tanpa bit *fraction* systemC

Flow chart model fungsional systemC ditunjukkan oleh gambar 6.14. Sedangkan *flow chart* implementasi algoritma Max-log-MAP dan SWA ditunjukkan oleh gambar 6.15. Proses perhitungan $B_k(s)$ dilakukan dalam jumlah 6 *clock* seperti yang diilustrasikan gambar 5.11. Selain itu, seperti yang diilustrasikan pada gambar 5.11 dan gambar 6.15 pengolahan subblok terakhir tidak dapat dilakukan secara paralel karena jumlah *clock* pemrosesannya berbeda dibanding subblok bukan terakhir dimana membutuhkan 6 *clock* untuk perhitungan $B_k(s)$ dan 5 *clock* untuk perhitungan $A_{k-1}(\acute{s})$ dan $L(u_k|y)$. Banyaknya *clock* untuk pengolahan subblok terakhir adalah sebanya $res = (K - (40 \times pckg\_div))/8$, dimana $pckg\_div = \lfloor K/40 \rfloor - 1$. Hal ini lebih dijelaskan oleh FSM yang ditunjukkan seperti gambar 6.16.

Gambar 6.14: *Flow chart* proses *decoding* top level arsitektur model fungsional turbo dekoder systemC

Gambar 6.15: *Flow chart* implementasi algoritma Max-log-MAP dan SWA model fungsional turbo dekoder systemC

Garis berwarna kuning pada gambar 6.16 menunjukkan aliran*state* yang dilalui untuk data dengan $K < 80$. Hal ini disebabkan untuk urutan data $K < 80$ data yang diterima merupakan subblok terakhir. Selain itu dalam gambar 6.16 terdapat dua komponen dekoder pertama. Perbedaan dari keduanya adalah komponen dekoder pertama yang berwarna biru lebih tua menandakan proses komponen dekoder pertama yang menyimpan dan sekaligus memproses input. Sedangkan yang kedua hanya memprosesnya. Berikut adalah deskripsi lebih lengkapnya untuk masing-masing *state*-nya.

- INIT :
  State ini merupakan *state* inisialisasi untuk semua *register* sehingga nilainya bernilai 0. *Trigger* utama untuk *state* ini untuk pindah ke *state* berikutnya adalah sinyal `start` dan `reset` yang menandakan bahwa proses *encoding* dimulai dan diasumsikan nilai K valid.

- FIRST_BCJRIN :
  Pada *state* ini turbo dekoder menyimpan input sekaligus menghitung $B_k(s)$. Proses ini berlangsung selama 6 kali yang artinya mendapatkan 48 *backward recursion metric*. Pada *state* BCJR1_FIRST dan BCJR2_FIRST dilakukan hal yang sama, namun perbedaannya adalah kedua *state* ini tidak menyimpan input, namun memanggil input yang telah disimpan.

- BCJRIN :
  State ini melakukan proses perhitungan paralel antara $B_k(s)$subblok berikutnya dengan $A_{k-1}(\acute{s})$ sekaligus dengan $L(u_k|y)$. Pada perhitungan $B_k(s)$ ini, input data disimpan sama dengan *state* sebelumnya. Proses ini berlangsung selama 6 *clock* untuk perhitungan $B_k(s)$dan 5 *clock*-nya untuk perhitungan $A_{k-1}(\acute{s})$ sekaligus dengan $L(u_k|y)$ dan proses ini berulang hingga subblok terakhir.

- BCJRIN_CLR_RES :
  State ini melakukan hal yang sama dengan *state* INIT. Begitu juga dengan *state* PRC_BCJR2_CLR_RES, dan PRC_BCJR1_CLR_RES.

- BCJRIN_PRC_RES :
  State ini melakukan perhitungan $B_k(s)$ untuk subblok terakhir. Begitu juga dengan *state* PRC_BCJR2_PRC_RES dan PRC_BCJR1_PRC_RES.

- BCJRIN_FIN_RES :
  State ini, *state* PRC_BCJR2_FIN_RES, dan PRC_BCJR1_FIN_RES melakukan perhitungan $A_{k-1}(\acute{s})$ dan $L(u_k|y)$ sekaligus untuk subblok terakhir.

- BCJR2_INT :
  State ini melakukan proses *interleaver* untuk subblok pertama yang akan diolah pada *state* berikutnya. Pada *state* BCJR1_INT proses serupa juga dilakukan, namun pada *state* ini proses *deinterleaver* yang dilakukan.

Implementasi persamaan 5.32, 5.33, dan 5.34 menjadi *datapath* tampak pada gambar 6.17. Pada gambar diilustrasikan juga lebar bus untuk *backward recursion metric* adalah 8 karena untuk tiap *state*-nya terdapat 8 $B_k(s)$. Blok *dual bank ram* digunakan

untuk mengatasi proses paralelisasi antara perhitungan $B_k(s)$ *state* berikutnya dengan perhitungan $A_{k-1}(\acute{s})$ dan $L(u_k|y)$. Hal ini sama seperti yang dilakukan pada model turbo enkoder sebelumnya. Sedangkan besar ukuran memorinya bergantung pada ukuran subblok terbesar, yaitu 72. Hal ini didapatkan pada pengolahan subblok terakhir. Program deklarasi memori untuk blok *dual bank ram* ditunjukkan pada program 6.9. Sehingga ukuran memorinya adalah $2 \times 72 \times 13$ bit $\times 8 = 14976$ bit ≈ 16 KB.

Program 6.9: Program Deklarasi Memori Implementasi Algoritma Max-log-MAP dan SWA pada SytemC

```
#define  WORD_BETA 13
#define  INTEGER_BETA 13
#define  RAM_DEPTH 72

SC_MODULE (dual_bank_ram_ar_sw) {
 ...

sc_fixed<WORD_BETA,INTEGER_BETA,SC_RND_MIN_INF,SC_SAT>
mem0 [RAM_DEPTH][8];
sc_fixed<WORD_BETA,INTEGER_BETA,SC_RND_MIN_INF,SC_SAT>
mem1 [RAM_DEPTH][8];

 ...
```

Gambar 6.16: FSM model fungsional turbo dekoder systemC

Gambar 6.17: Submodul gambar 5.12 untuk model RTL systemC

Implementasi *fixed point* pada systemC di atas disesuaikan dengan model MATLAB untuk kepentingan verifikasi. Parameter `SC_RND_MIN_INF` mengindikasikan mode kuantisasi yang digunakan adalah pembulatan menuju tak hingga. Sedangkan parameter `SC_SAT` mode saturasinya adalah saturasi. Parameter ini adalah pengaturan standar pada fungsi *fixed point* MATLAB.

Metode verifikasi yang digunakan sama dengan model fungsional pada turbo enkoder, namun perbedaannya adalah format data yang digunakan *fixed point*. Titik *corner case* yang digunakan adalah untuk ukuran $K = 6144$, $K = 40$, dan titik *average case*-nya adalah $K = 4480$. Indikasi bahwa model systemC ini identik dengan model MATLAB adalah variabel $benar$ bernilai $K/8$ dengan nilai $salah = 0$. Gambar hasil verifikasinya ditunjukkan pada gambar 6.19, 6.20, dan 6.21.

Gambar 6.18: *Platform* verifikasi model fungsional tubo dekoder systemC

Gambar 6.19: Hasil simulasi titik *corner case* ($K = 40$) model fungsional turbo dekoder systemC

Hasil yang didapat pada ketiga gambar hasil simulasi di atas (gambar 6.19, 6.20, dan 6.21) mengindikasikan bahwa model yang dibuat identik dengan model MATLAB. Selain itu, pada ketiga gambar tersebut dapat dilihat juga total *clock* yang dibutuhkan untuk proses *decoding*-nya (*clock budget*). Perhitungannya dirumuskan dalam persamaan 6.1.

$$\begin{aligned}
ctr_normal = &\ 6, \\
pckg_div = &\ \lfloor K/40 \rfloor - 1, \\
res = &\ (K - (40 \times pckg_div))/8, \\
clk_budget_input = &\ ctr_normal + (6 \times pckg_div) + res, \\
clk_budget_rsc = &\ (2 \times ctr_normal) + (6 \times pckg_div) + \\
&\ (2 \times res), \\
clk_budget = &\ clk_budget_input + \\
&\ 7 \times clk_budget_rsc + \\
&\ K/8 + FSM_budget.
\end{aligned} \tag{6.1}$$

Variabel clk_budget_input mengillustrasikan proses yang dilakukan oleh komponen dekoder pertama yang menyimpan input (lihat gambar 6.16 bagian biru lebih tua). Sedangkan FSM_budget mengillustrasikan banyaknya *clock* yang dibutuhkan untuk skenario tes yang dilakukan. Untuk analisis lebih lengkapnya akan dijabarkan pada bab berikutnya. Pada bab ini dapat disimpulkan bahwa Model fungsional RTL pada systemC baik turbo enkoder dan dekoder telah siap untuk dilakukan proses translasi ke HDL (bab selanjutnya).

Gambar 6.20: Hasil simulasi titik *corner case* ($K = 6144$) model fungsional turbo dekoder systemC

Gambar 6.21: Hasil simulasi titik *average case* ($K = 4480$) model fungsional turbo dekoder systemC

BAB 7

PERANCANGAN HDL DENGAN VERIFIKASI BERBASIS *ASSERTION*

Pada bab ini akan dilakukan proses transisi model fungsional yang telah dirancang pada bab sebelumnya menjadi HDL dengan menggunakan bahasa verilog. Bersamaan dengan perancangan ini akan dirancang pula *platform* verifikasinya menggunakan metode ABV dengan model fungsional systemC sebagai referensinya. Untuk implementasi masing-masing komponen pada gambar 4.2 digunakan bahasa pemograman yang berbeda seperti yang terlihat pada gambar 7.1. *Platform* ini diimplementasikan dalam modul *testbench* dimana *platform* ini dijalankan berulang kali hingga mencapai tingkat keyakinan tertentu (*LEVEL OF CONFIDENCE*).

Gambar 7.1: *Platform* verifikasi berbasis *assertion* dan deskripsi penggunaan bahasa pemogramannya masing-masing

7.1. Perancangan HDL

Prose perancangan HDL menggunakan verilog dengan model fungsional RTL systemC tidak terlalu rumit karena selain akurasi waktu dan portnya dirancang sedekat mungkin dengan implementasi HW, verilog memiliki tingkat kesamaan *syntax* yang hampir mirip dengan bahasa C [**?**]. Model fungsional systemC yang dirancang sebelumnya dirancang sehingga operator yang digunakan menggunakan *syntax* bahasa C.

Beberapa hal yang perlu diperhatikan pada proses translasi model fungsional kode turbo ke HDL, selain yang telah dijelaskan pada tabel 3.1, adalah operator `max` dan implementasi FSM. Sehingga, pada bagian ini akan dijelaskan lebih rinci mengenai perancangan operator `max` dan FSM.

7.1.1. Operator `max`

Permasalah yang didapat pada perancangan operator `max` ini adalah representasi data pada penggunaan *statement conditional*. Program 7.1 menunjukkan implementasinya

pada systemC. Sedangkan jika diimplementasikan secara langsung pada verilog, maka hasil yang didapat tidak sesuai harapan karena representasi data pada verilog pada *statement* yang demikian adalah *unsigned*.

Program 7.1: Program Implementasi Operator `max` pada SystemC

```
beta1_8k= (branch1_beta1_8k > branch2_beta1_8k) ? branch1_beta1_8k :
branch2_beta1_8k;
```

Program 7.2 menunjukkan implementasi operator `max` pada verilog. Langkah pertama yang harus dilakukan adalan memeriksa apakah variabel tersebut bertanda positif atau negatif. Hal ini ditunjukkan oleh ekpresi kondisional pada program di bawah. Jika *Most Significant Bit* (MSB) variable tersebut bernilai satu maka variabel tersebut bernilai negatif dan sebaliknya. Kemudian langkah selanjutnya adalah proses *assignment*.

Program 7.2: Program Implementasi Operator `max` pada Verilog

```
if (branch1_beta1_8k[INTEGER_BETA-1] == 1 &&
branch2_beta1_8k[INTEGER_BETA-1] == 0)
        beta1_8k = branch2_beta1_8k;
else if (branch1_beta1_8k[INTEGER_BETA-1] == 0 &&
branch2_beta1_8k[INTEGER_BETA-1] == 1)
        beta1_8k = branch1_beta1_8k;
else
        beta1_8k= (branch1_beta1_8k > branch2_beta1_8k) ?
        branch1_beta1_8k : branch2_beta1_8k;
```

7.1.2. Proses Translasi FSM ke HDL

Permasalahan yang didapat pada proses translasi FSM ke HDL adalah deklarasi nama *state* dan insisialisasi *state*. Pada model systemC digunakan tipe data enumerasi untuk deklarasi nama *state* seperti tampak pada program 7.3. Namun verilog tidak menyediakan tipe data enumerasi, oleh karena itu digunakan deklarasi parameter sebagai gantinya. Hasil implementasi dalam verilog tampak pada program 7.4.

Program 7.3: Program Deklarasi Nama *State* pada SystemC

```
enum typeState {MULAI, PREOLAH_INIT, OLAH_INIT,
                PREOLAH_NEXT_SEQ, OLAH_NEXT_SEQ,
                COMPARE,
                PREOLAH_PREFINISHING, OLAH_PREFINISHING,
                PREOLAH_FINISHING, OLAH_FINISHING
                  };
    typeState CurrentState , InitialState , NextState;
```

Program 7.4: Program Deklarasi Nama *State* pada Verilog

```
parameter MULAI                  = 0;
parameter PREOLAH_INIT           = 1;
parameter OLAH_INIT              = 2;
parameter PREOLAH_NEXT_SEQ = 3;
parameter OLAH_NEXT_SEQ          = 4;
parameter COMPARE                = 5;
parameter PREOLAH_PREFINISHING = 6;
parameter OLAH_PREFINISHING = 7;
parameter PREOLAH_FINISHING = 8;
parameter OLAH_FINISHING         = 9;

 reg  [3:0]  CurrentState ;
 reg  [3:0]   InitialState  ;
 reg  [3:0]  NextState ;
```

Permasalahan selanjutnya dalam proses translasi ini adalah insisialisasi FSM. Inisialisasi *state* FSM pada model systemC adalah *state* MULAI, namun hal ini tidak berlaku untuk verilog karena nilai awal dari sebuah variable pada verilog adalah X (*DON'T CARE*). Untuk mengatasinya maka ditambahkan program 7.5.

Program 7.5: Program Inisialisasi *State* pada Verilog

```
if  ( reset_in )  CurrentState  = MULAI;
else  CurrentState  = NextState ;
```

7.2. Perancangan Verifikasi Berbasis *Assertion*

Perancangan ABV ini dirancang bersamaan dengan proses translasi ke HDL. Perancangan ABV ini dibagi menjadi tiga, yaitu perancangan properti *assertion*, perancangan random input, dan perancangan *functional coverage*.

7.2.1. Perancangan Properti *Assertion*

Jenis properti yang dirancang pada tugas akhir ini adalah *counter*, FSM, dan sinyal konrol. Ketiga bagian tersebut sangat berpengaruh terhadap kinerja sistem turbo kode secara keseluruhan.

Properti *Assertion Counter*

Properti ini berguna untuk memeriksa nilai dari blok *counter*. Implementasi properti ini sangat dibutuhkan pada perancangan turbo kode khususnya pada turbo dekoder karena fungsi blok *counter* sangat essensial yaitu menentukan nilai alamat penyimpanan sebuah variabel di memori.

1. *Counter* untuk alamat *dual bank ram* pada turbo enkoder. Properti ini juga berlaku untuk indeks *generator* pada blok *interleaver* (lihat persamaan 5.27) dan *counter* lain yang menyimpan input.
   Properti *assertion* : nilai *counter* selalu kurang dari 768 ($6144/8$, dimana 6144 adalah K maksimum sesuai dengan standar LTE [**?**]).

```
property p_second_addr_counter;
  @(posedge clock) disable iff (reset) (count < 768);
endproperty
a_second_addr_counter : assert property(p_second_addr_counter);
```

2. *Counter* untuk alamat memori yang menyimpan *backward recursion metric*.
   Properti *assertion* : nilai *counter* selalu kurang dari 9 (*(Ukuran subblok terbesar)/8*).

```
property p_addr_counter_bcjread;
  @(posedge clock) disable iff (reset) (count < 9);
endproperty
a_addr_counter_bcjread : assert property(p_addr_counter_bcjread);
```

Properti *Assertion* FSM

Properti ini mengacu pada gambar FSM turbo enkoder (gambar 6.8) dan dekoder (gambar 6.16). Kegunaan utama properti ini adalah memeriksa kebenaran transisi antar *state*. Implementasi properti ini sebagian besar identik. Sehingga akan dijabarkan bebrapa properti saja.

1. Properti ini untuk memeriksa *state* yang boleh dilalui pada sebuah FSM.
   Properti *assertion* : *State-state* yang boleh dilalui pada FSM turbo enkoder (gambar 6.8) adalah MULAI, PREOLAH_INIT,OLAH_INIT, COMPARE, PREOLAH_PREFINISHING, OLAH_PREFINISHING, PREOLAH_FINISHING, OLAH_FINISHING, PREOLAH_NEXT_SEQ, O- LAH_NEXT_SEQ.

```
property p_fsm_trans_legal;
@(posedge clock) disable iff (reset)
     ( CurrentState inside{
        MULAI, PREOLAH_INIT, OLAH_INIT,
        COMPARE, PREOLAH_PREFINISHING,
        OLAH_PREFINISHING, PREOLAH_FINISHING,
        OLAH_FINISHING, PREOLAH_NEXT_SEQ,
        OLAH_NEXT_SEQ}
     );
endproperty
a_fsm_trans_legal : assert property(p_fsm_trans_legal);
```

2. Properti ini untuk memeriksa transisi untuk tiap *state* yang boleh dituju pada sebuah FSM.
   Properti *assertion* : Dari *state* MULAI, *state* berikutnya hanya boleh *state* MULAI atau PREOLAH_INIT.

```
property p_fsm_mulai_legal;
@(posedge clock) disable iff (reset)
    (CurrentState == MULAI) |=>
       (CurrentState == MULAI ||
       CurrentState == PREOLAH_INIT );
endproperty
a_fsm_mulai_legal : assert property(p_fsm_mulai_legal);
```

Properti *Assertion* Sinyal Kontrol

Sama halnya dengan properti *assertion counter*, properti *assertion* sinyal kontrol juga penting karena sinyal kontrol inilah yang mengatur kerja blok *datapath*. Berikut adalah beberapa properti *assertion* yang dirancang pada tugas akhir ini.

1. Properti ini untuk memeriksa sinyal kontrol pemilihan memori yang diakses pada *dual bank ram* agar tidak terjadi kesalahan penyimpanan atau pengmbilan data.
   Properti *assertion* : Sinyal `bank_r` selalu tidak boleh sama dengan sinyal `bank_w`.

```
property p_bank_ram;
    @(posedge clock) disable iff (reset)
     (bank_r == !bank_w);
endproperty
```

```
a_bank_ram : assert property(p_bank_ram);
```

2. Properti ini untuk memeriksa sinyal kontrol *enable* pada memori.
   Properti *assertion* : Sinyal `cs` harus aktif jika memori sedang ditulis atau dibaca.

```
property p_control_ram;
    @(posedge clock) disable iff (reset)
     (we || oe) |-> cs;
endproperty
a_control_ram : assert property(p_control_ram);
```

3. Properti ini untuk memeriksa sinyal kontrol pengaktifan blok *interleaver* dan konvolusional enkoder pada turbo enkoder.
   Properti *assertion* : Sinyal `conv_en` harus aktif jika sinyal `int_en` aktif.

```
property p_conv_int_en;
    @(posedge clock) disable iff (reset)
    (conv_en) |-> int_en;
endproperty
a_conv_int_en : assert property(p_conv_int_en);
```

4. Properti ini untuk memeriksa sinyal kontrol valid hubungan antara bit *tail* dan sinyal valid output.
   Properti *assertion* : Sinyal `out_valid` harus aktif jika sinyal `tail_valid` aktif.

```
property p_tail_out_valid;
    @(posedge clock) disable iff (reset)
    (tail_valid) |-> out_valid;
endproperty
a_tail_out_valid : assert property(p_tail_out_valid);
```

7.2.2. Perancangan Random Input

Selain dapat mengimplementasikan random input untuk urutan data bitnya, pada tugas akhir ini pemilihan ukuran data K dapat juga diacak. Tipe randomisasi untuk pemilihan K ini adalah *constrained randomization* yang artinya data random tersebut dapat dibatasi jangkauan nilainya.

Seperti yang dilihat pada tabel 2.1, nilai K bukan merupakan urutan dengan beda 1. Sehingga diperlukan sebuah *array* untuk menyimpan nilai K tersebut dan indeksnya lah yang diakses secara random. Batasan random indeks tersebut adalah dari 0 sampai 187. Program 7.6 menunjukkan deklarasi *constrained randomization* yang telah dirancang.

Program 7.6: Program Deklarasi *Constrained Randomization*

```
scv_smart_ptr  <sc_uint<13> > random_panjang_in1;
 ...

void  testbench  ()  {
     random_panjang_in1->keep_only(0,187);

 ...
```

Program di bawah adalah potongan program implementasi random input pada modul *testbench*. Variabel `parameter` adalah *array* yang menyimpan semua nilai K pada tabel 2.1.

Program 7.7: Program Implementasi Random Input

```
scv_smart_ptr  <sc_uint<13> > random_panjang_in1;
scv_smart_ptr  <bool> random_data_in;
 ...

void  testbench  ()  {
     random_panjang_in1->next();
     temp_random_panjang_in1 = *random_panjang_in1;
 ...
     temp_PANJANG_INPUT1 = parameter[temp_random_panjang_in1] ;
 ...
     random_data_in->next();
     data_in0 = *random_data_in;
 ...
```

7.2.3. Perancangan *Functional Coverage*

Functional coverage digunakan untuk mengetahui seberapa besar implementasi desain sesuai dengan spesifikasi yang diharapkan. *Functional coverage* yang bernilai 100% berarti bahwa semua spesifikasi yang diharapkan telah terpenuhi. Pada SVA, *functional coverage* dapat diimplementasikan dengan menggunakan fungsi `cover property<nama_property>`. Pada tugas akhir ini, setiap properti *assertion* yang terpenuhi, maka akan ada variable integer yang digunakan untuk menghitung banyak-

nya *functional coverage* tersebut terpenuhi. Contoh implementasi *functional coverage* pada properti yang telah dibuat ditunjukkan pada program di bawah.

Program 7.8: Program Implementasi *Functional Coverage*

```
c_index_int_counter : cover property (p_index_int_counter);
```

BAB 8

HASIL SIMULASI HARDWARE, SINTESIS, DAN ANALISIS KERJA

Pada bab ini akan ditampilkan hasil simulasi perancangan HDL dan ABV pada bab sebelumnya dan hasil sintesisnya. Pertama-tama akan ditampilkan hasil simulasinya. Pada bab ini juga akan dijabarkan perbandingan optimasi pada blok konvolusional enkoder dan indeks *generator* pada blok inter yang telah dijabarkan pada bab 5. Perbandingna ini hanya dilakukan untuk blok tersebut karena kedua blok itulah yang terdapat optimasi paralelisasi yang dilakukan. Kemudia dari kedua hasil simulasi dan sintesisnya, akan dianalisis apakah hasilnya sesuai yang diharapkan. Perancangan dinyatakan benar jika hasil implementasi HW sesuai dengan model fungsional, dan *functional coverage* terpenuhi semuanya yang artinya bernilai 100%. Selain itu, hasil sintesisnya memenuhi tujuan dari tugas akhir ini.

8.1. Perbandingan Hasil Optimasi Konvolusional Enkoder dan Indeks *Generator* pada Blok *Interleaver*

Kedua tabel di bawah 8.1 dan 8.2 merupakan rangkuman perbandingan dari optimasi yang telah dilakukan pada konvolusional enkoder dan indeks textitgenerator pada blok inter. Terdapat 3 jenis arsitektur yang dibandingkan, yaitu *unparallized*, *unoptimized*, dan *optimized*. Arsitektur *unparallized* mengacu kepada persamaan aslinya (persamaan 5.18 untuk konvolusional enkoder dan persamaan 2.8 untuk *interleaver*). Arsitektur *unoptimized* mengacu kepada persamaan aslinya (persamaan 5.24 untuk konvolusional enkoder dan persamaan 5.26 untuk *interleaver*). Sedangkan arsitektur *unoptimized* mengacu kepada persamaan aslinya (persamaan 5.25 untuk konvolusional enkoder dan persamaan 5.27 untuk *interleaver*).

Optimasi yang ditunjukkan pada tabel 8.1 adalah pengurangan area operator `xor` dengan hanya penalti 1 blok `xor` dibandingkan arsitektur *unparallized*. Sedangkan untuk blok *interleaver* terdapat pengurangan 1 blok multiplier dengan penalti 8 blok adder. Namun dari segi area jumlah blok multiplier dikurangin hingga hanya tersisa 1 multiplier dibandingkan 24 multiplier pada arsitektur *unparallized*. Untuk hasil lengkapnya

dapat dilihat pada kedua tabel di bawah.

Tabel 8.1: Perbandingan *critical path* dan area konvolusional enkoder

Arsitektur	***Critical Path***	**XOR**
Unparallized	$3T_{xor}$	8
Unoptimized	$3T_{xor}$	48
Optimized	$4T_{xor}$	27

Tabel 8.2: Perbandingan *critical path* dan area indeks *generator* blok *interleaver*

Arsitektur	***Critical Path***	**Adder**	**Multiplier**	**Modulo**	**Shifter** 3	**ROM**
Unparallized	$2T_{mult} + T_{add} + T_{mod}$	1	3	1	0	0
Unoptimized	$T_{shift} + 2T_{mult} + T_{add} + T_{mod}$	22	24	8	1	1
Optimized	$T_{mult} + 9T_{add} + T_{mod}$	16	1	8	1	1

Jika dibandingkan kedua tabel tersebut maka *critical path* sistem terletak pada blok *interleaver*. Gambar 5.9 mengillustrasikan jalur tersebut yang ditandai warna oranye.

8.2. Hasil dan Analisis Simulasi Hardware

8.2.1. Hasil dan Analisis Simulasi Turbo Enkoder

Gambar 8.1 menunjukkan hasil simulasi HW yang telah dirancang. Sedangkan pada gambar 8.3 menunjukkan hasil implementasi *functional coverage*-nya dan teks di bawah merupakan laporan untuk implementasi tersebut. Dari segi kebenaran seperti yang ditunjukkan pada gambar 8.1, variabel $salah$ bernilai 0 dan variabel $benar$ memiliki sebuah nilai (menunjukkan berapa kali pengecekan) yang berarti untuk semua perbandingan nilai yang dihasilkan HW turbo enkoder identik dengan model fungsionalnya. Hasil lebih lengkapnya untuk semua percobaan ditampilkan pada lampiran A.

Oleh karena hasil simulasinya benar, maka dapat dipastikan bahwa implementasi *functional coverage*-nya juga benar semua. Hal ini ditunjukkan seperti pada gambar 8.3. Pda gambar tersebut semua implementasi properti *assertion*-nya benar. Laporan lengkapnya ditunjukkan teks di bawah. Pada akhir laporan tersebut dinyatakan bahwa `TOTAL DIRECTIVE COVERAGE: 100.0% COVERS: 18` yang artinya terdapat 18 jenis properti *assertion* dan semua properti tersebut terpenuhi.

Pada metode ABV juga harus dipastikan bahwa titik tes *corner*-nya juga harus diverifikasi. Gambar 8.2 menunjukkan hasil simulasi versi *corner test*. Nilai `benar` besarnya sama yang didapat pada verifikasi model fungsional systemC terhadap MATLAB seperti yang ditunjukkan pada gambar 6.11. Hasil *functional coverage* dapat dilihat

pada gambar 8.4. Dapat dilihat pada gambar tersebut bahwa banyaknya properti *assertion* yang terpenuhi lebih sedikit. Hal ini ditunjukkan oleh variabel `count`. Besar `TOTAL DIRECTIVE COVERAGE dan COVERS`-nya pun sama dengan yang didapat pada teks di bawah. Sehingga dari hasil yang didapat dapat disimpulkan bahwa secara fungsional rancangan HDL ini benar.

```
DIRECTIVE COVERAGE:
---------------------------------------------------------------------------
Name    Design Design   Lang File(Line) Count Status
        Unit   UnitType
---------------------------------------------------------------------------
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_trans_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(873)206033 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_preolah_fin_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(874)  200 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_mulai_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(875)  200 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_compare_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(876)  300 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_preolah_init_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(877)  500 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_olah_init_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(878)47323 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_olah_nextseq_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(879)36329 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_preolah_nextseq_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(880)  200 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_preolah_prefin_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(881)  500 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_olah_prefin_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(882)72658 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_fsm_olah_fin_legal
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(883)47123 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_bank_ram
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(884)206033 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_control_ram
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(885)203933 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_conv_int_en
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(886)156410 Covered
/test/RTL_turbo_encoder_INST/fsm/genblk1/c_tail_out_valid
       FSM_enc Verilog  SVA  RTL/FSM_enc.v(887)  200 Covered
/test/RTL_turbo_encoder_INST/datapath/comb/first/count_1st/genblk1/
   c_first_addr_counter
       counter_8level_1st Verilog  SVA  RTL/
       counter_8level_1st.v(73)206033 Covered
/test/RTL_turbo_encoder_INST/datapath/comb/second/int_ram/inter/count/genblk1/
   c_index_int_counter
       counter_j Verilog  SVA  RTL/counter_j.v(32)206033 Covered
/test/RTL_turbo_encoder_INST/datapath/comb/second/int_ram/count/genblk1/
    c_second_addr_counter
       counter_8level Verilog  SVA  RTL/counter_8level.v(73)206033 Covered
```

```
TOTAL DIRECTIVE COVERAGE: 100.0%  COVERS: 18
```

8.2.2. Hasil dan Analisis Simulasi Turbo Dekoder

Sama halnya dengan hasil simulasi turbo enkoder di atas. Gambar 8.1 menunjukkan hasil simulasi implementasi HW dan ABV. Pada gambar tersebut ditunjukkan bahwa nilai `salah`-nya bernilai 0 dengan nilai `benar` menunjukkan sebuah angka yang berarti tidak ada kesalahan dan hasilnya sama dengan model fungsional. Hasil lebih lengkapnya untuk semua percobaan ditampilkan pada lampiran A.

Untuk hasil implementasi *functional coverage*-nya yang ditunjukkan oleh teks di bawah menunjukkan bahwa semua properti *assertion* terpenuhi dengan laporan nilai `TOTAL DIRECTIVE COVERAGE: 100.0% COVERS: 54`. Titik *corner case* ditunjukkan oleh gambar 8.6 dan 8.7. Nilai `benar` yang didapat sama dengan nilai `benar` pada gambar 6.19 dan 6.20 yaitu gambar hasil simulasi model fungsional turbo dekoder. Namun, untuk implementasi *functional coverage* titik *corner case* $K = 40$ yang ditunjukkan pada gambar 8.9 mengillustrasikan bahwa tidak semua properti *assertion* terpenuhi karena jika ditinjau dari FSM-nya (gambar 6.16) pada kasus $K < 80$ tidak semua *state* terlewati. Bagaimanapun juga hal ini tidak mengindikasikan bahwa rancangan yang telah dibuat salah. Benar atau salah sebuah rancangan jika menggunakan metode ABV ini adalah tidak ada properti *assertion* yang salah. Sehingga dari hasil yang didapat dapat disimpulkan bahwa secara fungsional rancangan HDL ini benar.

[htbp]

```
DIRECTIVE COVERAGE:
-------------------------------------------------------------------------------
Name            Design Design   Lang File(Line) Count Status
                Unit   UnitType
-------------------------------------------------------------------------------
/test/RTL_turbo_decoder_INST/datapath_bl/bcj/counter/genblk1/
  c_addr_counter_bcjread
       down_counter8 Verilog  SVA  RTL/down_counter8.v(80)127269 Covered
/test/RTL_turbo_decoder_INST/datapath_bl/bcj/upcounter/genblk1/
  c_addr_counter_bcjwrite
       up_counter8 Verilog  SVA  RTL/up_counter8.v(115)127269 Covered
/test/RTL_turbo_decoder_INST/datapath_bl/ramL/upcount/genblk1/
  c_addr_counter_Lwrite
       up_counterL Verilog  SVA  RTL/up_counterL.v(99)59632 Covered
/test/RTL_turbo_decoder_INST/datapath_bl/ramL/upcountout/genblk1/
  c_addr_counter_Lread
       up_counterL_out Verilog  SVA  RTL/up_counterL_out.v(78)66370 Covered
```

```
/test/RTL_turbo_decoder_INST/datapath_bl/int_ram/inter/count/genblk1/
  c_index_int_counter
       counter_j Verilog  SVA  RTL/counter_j.v(33)127567 Covered
/test/RTL_turbo_decoder_INST/datapath_bl/int_ram/count/genblk1/
  c_addr_counter_int
       counter_8level Verilog  SVA  RTL/counter_8level.v(68)127567 Covered
/test/RTL_turbo_decoder_INST/datapath_bl/deinter_ram/deinter/count/genblk1/
  c_index_deinter_counter
       counter_j_deinter Verilog  SVA  RTL/counter_j_deinter.v(33)112764 Covered
/test/RTL_turbo_decoder_INST/datapath_bl/deinter_ram/count/genblk1/
  c_addr_counter_deinter
       counter_8level_deinter Verilog  SVA  RTL/counter_8level_deinter.v(68)
       112764 Covered
...
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_fsm_prc_bcjr2_clr_res_legal
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2910)  200 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_fsm_prc_bcjr2_prc_res_legal
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2911) 1372 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_fsm_prc_bcjr2_fin_res_legal
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2912) 1372 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_fsm_out_valid_legal
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2913)11475 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_bcjr_bank
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2914)129928 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_bcjr_ram
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2915)127125 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_inter_ram
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2916)92584 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_deinter_ram
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2917)93610 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_input2_ram
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2918)56480 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_Wk_ram
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2919)43224 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_Wkup_ram
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2920)41985 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_input_ram
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2921)111416 Covered
/test/RTL_turbo_decoder_INST/FSM_dec_bl/genblk1/c_control_L_ram
       FSM_dec Verilog  SVA  RTL/FSM_dec.v(2922)65038 Covered

TOTAL DIRECTIVE COVERAGE: 100.0%  COVERS: 54
```

Gambar 8.1: Hasil simulasi implementasi HW dan ABV turbo enkoder

Gambar 8.2: Hasil simulasi implementasi HW dan ABV turbo enkoder titik *corner case* ($K_1 = K_2 = 6144$ dan $K_3 = 40$

Gambar 8.3: *Functional coverage platform* ABV pada turbo enkoder

Gambar 8.4: *Functional coverage platform* ABV pada turbo enkoder titik *corner case* ($K_1 = K_2 = 6144$ dan $K_3 = 40$

8.3. Hasil dan Analisis Sintesis Turbo Kode

Setelah perancangan HW hasilnya benar secara fungsional, maka HW tersebut disintesis untuk memastikan bahwa rancangan kita dapat disintesis dan memenuhi tujuan akhir pada perancangan tugas akhir ini, yaitu waktu latensi maksimum 0.1 ms dengan total *clock* yang dibutuhkan kurang dari 10000 *clock* (frekuensi minimum 100 MHz) untuk turbo dekoder. Untuk turbo enkoder di bawah 10000 *clock*dengan frekuensi minimum 100 MHz.

8.3.1. Hasil dan Analisis Sintesis Turbo Enkoder

Gambar 8.11 menyediakan laporan mengenai hasil sintesis untuk besar area, sedangkan gambar 8.13 menyediakan laporan mengenai hasil sintesis waktunya. Pada laporan area hal yang perlu diperhatikan adalah besarnya memori yang dibutuhkan. Hal ini karena implementasi blok *dual bank ram* pada turbo enkoder. Besarnya lebih dari 4×6144 (terdapat 2 blok *dual bank ram*).

Sedangkan untuk hasil sintesis waktunya ditunjukkan seperti pada gambar 8.12. Pada laporan tersebut menyediakan 4 informasi waktu yaitu, t_{su} (waktu *setup*), t_{co} (waktu *clock to output*), dan t_h (waktu *hold*). Dan frekuensi kerja yang diijinkan adalah 115.85 MHz. Frekuensi ini ditentukan dari waktu *critical path*. Seperti yang ditunjukkan pada gambar tersebut bahwa *critical path*-nya terletak pada blok indeks *generator interleaver*. Hal ini sesuai dengan hasil analisis perbandingan yang ditunjukkan gambar 5.9 yaitu dari sinyal `K` ke sinyal `pi_8jplus7[11]`. Selain itu perlu diperhatikan bahwa hasil sintesis waktunya benar karena ketiga waktu tersebut lebih kecil dibanding waktu *critical path*.

Gambar 8.5: Hasil simulasi implementasi HW dan ABV turbo dekoder

Gambar 8.6: Hasil simulasi implementasi HW dan ABV turbo dekoder titik *corner case* ($K = 40$)

Gambar 8.7: Hasil simulasi implementasi HW dan ABV turbo dekoder titik *corner case* ($K = 6144$)

Gambar 8.8: *Functional coverage platform* ABV pada turbo dekoder

Gambar 8.9: *Functional coverage platform* ABV pada turbo dekoder titik *corner case* ($K = 40$)

Gambar 8.10: *Functional coverage platform* ABV pada turbo dekoder titik *corner case* ($K = 6144$)

8.3.2. Hasil dan Analisis Sintesis Turbo Dekoder

Pada laporan sintesis waktu (gambar 8.14) turbo dekoder hasilnya sama dengan yang didapat pada turbo enkoder. Hal ini menyimpulkan bahwa meskipun diaplikasikan di blok turbo dekoder, *interleaver* ini tetap menjadi bagian dari *critical path*-nya. Sehingga dari segi frekuensi yang diijinkan rancangan turbo dekoder masih memenuhi tujuan perancangan tugas akhir ini yaitu (frekuensi minimumnya 100 Mhz).

Sedangkan untuk laporan area (gambar 8.13) menunjukkan pembengkakan pada ukuran memorinya. Hal ini disebabkan karena turbo dekoder banyak membutuhkan memori, baik untuk blok *interleaver*, *deinterleaver*, menyimpan input, dan untuk menyimpan *backward recursion metric*.

```
+-------------------------------------------------------------------------------+
; Flow Summary                                                                  ;
+-----------------------------------+-------------------------------------------+
; Flow Status                       ; Successful - Wed Jul 27 11:24:01 2011   ;
; Quartus II Version                ; 9.0 Build 132 02/25/2009 SJ Full Version;
dercoder; Revision Name             ; RTL_turbo_encoder                        ;
dercoder; Top-level Entity Name     ; RTL_turbo_encoder                        ;
; Family                           ; Cyclone II                                ;
; Device                           ; EP2C35F672C6                              ;
; Timing Models                    ; Final                                     ;
; Met timing requirements          ; Yes                                       ;
; Total logic elements             ; 1,268 / 33,216 ( 5 %)                     ;
;     Total combinational functions ; 1,103 / 33,216 ( 4 % )                   ;
;     Dedicated logic registers     ; 165 / 33,216 ( < 1 % )                    ;
; Total registers                  ; 165                                       ;
; Total pins                       ; 87 / 475 ( 18 % )                         ;
; Total virtual pins               ; 0                                         ;
; Total memory bits                ; 25,171 / 483,840 ( 5 % )                  ;
; Embedded Multiplier 9-bit elements ; 0 / 70 ( 0 % )                          ;
; Total PLLs                       ; 0 / 4 ( 0 % )                             ;
+-----------------------------------+-------------------------------------------+
```

Gambar 8.11: Laporan hasil sintesis area turbo enkoder

```
--------------------------------------------------------------------------------
Timing Analyzer Summary
--------------------------------------------------------------------------------

Type           : Worst-case tsu
Slack          : N/A
Required Time  : None
Actual Time    : 5.547 ns
From           : K[5]
To             : index_gen_int:inter|pi_8jplus7[11]
From Clock     : --
To Clock       : clock
Failed Paths   : 0

Type           : Worst-case tco
Slack          : N/A
Required Time  : None
Actual Time    : 6.983 ns
From           : index_gen_int:inter|counter_j:count|count[4]
To             : pi_8jplus7[11]
From Clock     : clock
To Clock       : --
Failed Paths   : 0

Type           : Worst-case th
Slack          : N/A
Required Time  : None
Actual Time    : 0.341 ns
From           : int_en
To             : index_gen_int:inter|counter_j:count|count[0]
From Clock     : --
To Clock       : clock
Failed Paths   : 0

Type           : Clock Setup: 'clock'
Slack          : N/A
Required Time  : None
Actual Time    : 115.85 MHz ( period = 8.632 ns )
From           : index_gen_int:inter|counter_j:count|count[4]
To             : index_gen_int:inter|pi_8jplus7[11]
From Clock     : clock
To Clock       : clock
Failed Paths   : 0

Type           : Total number of failed paths
Slack          :
Required Time  :
Actual Time    :
From           :
To             :
From Clock     :
To Clock       :
Failed Paths   : 0

--------------------------------------------------------------------------------
```

Gambar 8.12: Laporan hasil sintesis waktu turbo enkoder

```
+-------------------------------------------------------------------------------+
; Flow Summary                                                                  ;
+-----------------------------------+-------------------------------------------+
; Flow Status                       ; Successful - Wed Jul 27 11:50:11 2011   ;
; Quartus II Version                ; 9.0 Build 132 02/25/2009 SJ Full Version;
dercoder; Revision Name             ; RTL_turbo_decoder                        ;
dercoder; Top-level Entity Name     ; RTL_turbo_decoder                        ;
; Family                           ; Cyclone II                                ;
; Device                           ; EP2C35F672C6                              ;
; Timing Models                    ; Final                                     ;
; Met timing requirements          ; Yes                                       ;
; Total logic elements             ; 23,422 / 33,216 ( 71 %)                   ;
;     Total combinational functions ; 22,873 / 33,216 ( 69 % )                 ;
;     Dedicated logic registers     ; 549 / 33,216 ( 2 % )                     ;
; Total registers                  ; 549                                       ;
; Total pins                       ; 369 / 475 ( 77 % )                        ;
; Total virtual pins               ; 0                                         ;
; Total memory bits                ; 280,787 / 483,840 ( 58 % )                ;
; Embedded Multiplier 9-bit elements ; 0 / 70 ( 0 % )                          ;
; Total PLLs                       ; 0 / 4 ( 0 % )                             ;
+-----------------------------------+-------------------------------------------+
```

Gambar 8.13: Laporan hasil sintesis area turbo dekoder

```
--------------------------------------------------------------------------------
Timing Analyzer Summary
--------------------------------------------------------------------------------

Type           : Worst-case tsu
Slack          : N/A
Required Time  : None
Actual Time    : 5.547 ns
From           : K[5]
To             : index_gen_int:inter|pi_8jplus7[11]
From Clock     : --
To Clock       : clock
Failed Paths   : 0

Type           : Worst-case tco
Slack          : N/A
Required Time  : None
Actual Time    : 6.983 ns
From           : index_gen_int:inter|counter_j:count|count[4]
To             : pi_8jplus7[11]
From Clock     : clock
To Clock       : --
Failed Paths   : 0

Type           : Worst-case th
Slack          : N/A
Required Time  : None
Actual Time    : 0.341 ns
From           : int_en
To             : index_gen_int:inter|counter_j:count|count[0]
From Clock     : --
To Clock       : clock
Failed Paths   : 0

Type           : Clock Setup: 'clock'
Slack          : N/A
Required Time  : None
Actual Time    : 115.85 MHz ( period = 8.632 ns )
From           : index_gen_int:inter|counter_j:count|count[4]
To             : index_gen_int:inter|pi_8jplus7[11]
From Clock     : clock
To Clock       : clock
Failed Paths   : 0

Type           : Total number of failed paths
Slack          :
Required Time  :
Actual Time    :
From           :
To             :
From Clock     :
To Clock       :
Failed Paths   : 0

--------------------------------------------------------------------------------
```

Gambar 8.14: Laporan hasil sintesis waktu turbo dekoder

BAB 9

KESIMPULAN DAN SARAN

Pada bab ini dipaparkan kesimpulan dan saran untuk tugas akhir yang telah dilakukan. Kesimpulan didapatkan dari hasil analisis bab-bab sebelumnya. Saran berisi masukan untuk pengembangan turbo kode ini.

9.1. Kesimpulan

1. Perancangan kode turbo memiliki total *clock* dengan skenario paling buruk adalah 8295 *clock* dan dengan frekuensi kerja yang diijinkan adalah 115.85 Mhz untuk turbo dekoder. Sedangkan untuk turbo enkoder dibutuhkan maksimal sekitar 768×2 *clock* dan dengan frekuensi kerja yang diijinkan adalah 115.85 Mhz.

2. Pada tugas akhir ini berhasil diimplementasikan pendekatan logaritma natural eksponensial dengan algoritma Jacobian yang mendekatinya dengan menggunakan operator yang lebih sederhana yaitu operator max dari pangkat eksponensial tersebut.

3. Ukuran memori maksimal yang dibutuhkan untuk implementasi HW komponen dekoder adalah 16 KB.

4. Pada tugas akhir ini telah berhasil diimplementasikan model fungsional RTL pada systemC.

5. Pada tugas akhir ini telah berhasil diterapkan metode verifikasi ABV dengan semua propertinya terpenuhi 100% (18 properti untuk turbo enkoder dan 54 properti untuk turbo dekoder).

6. HDL dapat disintesis pada FPGA Altera Cyclone II EP2C35F672C6.

9.2. Saran

1. Agar didapat waktu *critical path* yang lebih pendek sehingga frekuensi kerjanya semakin cepat maka implementasi operator adder dan modulo dapat digunakan

algoritma yang dapat mempercepat kerja kedua operator tersebut.

2. Jika keadaannya dibutuhkan pemeriksaan performma BER yang lebih mudah dilakukan di MATLAB sedangkan metode perancangan sebuah sistem menuntut penggunaan simulasi HW dan SW bersamaan yang hanya dapat diimplementasikan jika model fungsionalnya menggunakan bahasa C atau C++, maka lebih baik disiapkan *environment* yang mendukung, misalnya penggunaan phyton untuk eksekusi pengecekan performa BER dari model C atau C++ di MATLAB.

3. Jika *tool* yang mendukung proses sintesis langsung dari systemC sudah dapat menghasilkan hasil sintesis yang sama optimalnya jika disintesis langsung dari HDL, maka proses perancangan pada tugas akhir ini tidak perlu dilakukan implementasi HDLnya.

LAMPIRAN A

HASIL SIMULASI HW DAN ABV KODE TURBO (TRANSKRIP)

Hasil simulasi implementasi HW dan ABV turbo enkoder

```
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 1%
# PANJANG INPUT1 = 4480
# PANJANG INPUT2 = 4480
# PANJANG INPUT3 = 456
# BENAR = 1794
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 2%
# PANJANG INPUT1 = 192
# PANJANG INPUT2 = 192
# PANJANG INPUT3 = 184
# BENAR = 1912
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 3%
# PANJANG INPUT1 = 896
# PANJANG INPUT2 = 896
# PANJANG INPUT3 = 160
# BENAR = 2288
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 4%
# PANJANG INPUT1 = 5568
# PANJANG INPUT2 = 5568
# PANJANG INPUT3 = 1152
# BENAR = 4664
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 5%
# PANJANG INPUT1 = 2016
# PANJANG INPUT2 = 2016
# PANJANG INPUT3 = 336
# BENAR = 5504
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
```

```
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 6%
# PANJANG INPUT1 = 576
# PANJANG INPUT2 = 576
# PANJANG INPUT3 = 208
# BENAR = 5772
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 7%
# PANJANG INPUT1 = 2624
# PANJANG INPUT2 = 2624
# PANJANG INPUT3 = 1440
# BENAR = 7116
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 8%
# PANJANG INPUT1 = 4480
# PANJANG INPUT2 = 4480
# PANJANG INPUT3 = 2624
# BENAR = 9452
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 9%
# PANJANG INPUT1 = 5504
# PANJANG INPUT2 = 5504
# PANJANG INPUT3 = 928
# BENAR = 11748
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 10%
# PANJANG INPUT1 = 504
# PANJANG INPUT2 = 504
# PANJANG INPUT3 = 360
# BENAR = 12027
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 11%
# PANJANG INPUT1 = 160
# PANJANG INPUT2 = 160
# PANJANG INPUT3 = 104
# BENAR = 12113
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 12%
# PANJANG INPUT1 = 3904
# PANJANG INPUT2 = 3904
# PANJANG INPUT3 = 1440
# BENAR = 13937
# SALAH = 0
```

```
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 13%
# PANJANG INPUT1 = 896
# PANJANG INPUT2 = 896
# PANJANG INPUT3 = 48
# BENAR = 14285
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 14%
# PANJANG INPUT1 = 2496
# PANJANG INPUT2 = 2496
# PANJANG INPUT3 = 784
# BENAR = 15417
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 15%
# PANJANG INPUT1 = 2368
# PANJANG INPUT2 = 2368
# PANJANG INPUT3 = 2304
# BENAR = 16881
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 16%
# PANJANG INPUT1 = 5184
# PANJANG INPUT2 = 5184
# PANJANG INPUT3 = 3584
# BENAR = 19721
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 17%
# PANJANG INPUT1 = 1504
# PANJANG INPUT2 = 1504
# PANJANG INPUT3 = 144
# BENAR = 20321
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 18%
# PANJANG INPUT1 = 2880
# PANJANG INPUT2 = 2880
# PANJANG INPUT3 = 1856
# BENAR = 21865
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 19%
# PANJANG INPUT1 = 3712
# PANJANG INPUT2 = 3712
# PANJANG INPUT3 = 672
# BENAR = 23425
```

```
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 20%
# PANJANG INPUT1 = 2432
# PANJANG INPUT2 = 2432
# PANJANG INPUT3 = 288
# BENAR = 24409
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 21%
# PANJANG INPUT1 = 3264
# PANJANG INPUT2 = 3264
# PANJANG INPUT3 = 376
# BENAR = 25727
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 22%
# PANJANG INPUT1 = 1632
# PANJANG INPUT2 = 1632
# PANJANG INPUT3 = 592
# BENAR = 26487
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 23%
# PANJANG INPUT1 = 4032
# PANJANG INPUT2 = 4032
# PANJANG INPUT3 = 456
# BENAR = 28113
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 24%
# PANJANG INPUT1 = 2496
# PANJANG INPUT2 = 2496
# PANJANG INPUT3 = 688
# BENAR = 29221
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 25%
# PANJANG INPUT1 = 3584
# PANJANG INPUT2 = 3584
# PANJANG INPUT3 = 800
# BENAR = 30765
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 26%
# PANJANG INPUT1 = 5824
# PANJANG INPUT2 = 5824
# PANJANG INPUT3 = 720
```

```
# BENAR = 33129
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 27%
# PANJANG INPUT1 = 264
# PANJANG INPUT2 = 264
# PANJANG INPUT3 = 144
# BENAR = 33264
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 28%
# PANJANG INPUT1 = 4800
# PANJANG INPUT2 = 4800
# PANJANG INPUT3 = 408
# BENAR = 35166
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 29%
# PANJANG INPUT1 = 5184
# PANJANG INPUT2 = 5184
# PANJANG INPUT3 = 1216
# BENAR = 37414
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 30%
# PANJANG INPUT1 = 5120
# PANJANG INPUT2 = 5120
# PANJANG INPUT3 = 1856
# BENAR = 39798
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 31%
# PANJANG INPUT1 = 5632
# PANJANG INPUT2 = 5632
# PANJANG INPUT3 = 512
# BENAR = 42038
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 32%
# PANJANG INPUT1 = 2016
# PANJANG INPUT2 = 2016
# PANJANG INPUT3 = 672
# BENAR = 42962
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 33%
# PANJANG INPUT1 = 672
# PANJANG INPUT2 = 672
```

```
# PANJANG INPUT3 = 400
# BENAR = 43314
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 34%
# PANJANG INPUT1 = 4160
# PANJANG INPUT2 = 4160
# PANJANG INPUT3 = 624
# BENAR = 45030
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 35%
# PANJANG INPUT1 = 3520
# PANJANG INPUT2 = 3520
# PANJANG INPUT3 = 1376
# BENAR = 46694
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 36%
# PANJANG INPUT1 = 6144
# PANJANG INPUT2 = 6144
# PANJANG INPUT3 = 120
# BENAR = 49028
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 37%
# PANJANG INPUT1 = 4608
# PANJANG INPUT2 = 4608
# PANJANG INPUT3 = 2048
# BENAR = 51268
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 38%
# PANJANG INPUT1 = 2112
# PANJANG INPUT2 = 2112
# PANJANG INPUT3 = 304
# BENAR = 52136
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 39%
# PANJANG INPUT1 = 1600
# PANJANG INPUT2 = 1600
# PANJANG INPUT3 = 320
# BENAR = 52816
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 40%
# PANJANG INPUT1 = 2880
```

```
# PANJANG INPUT2 = 2880
# PANJANG INPUT3 = 2688
# BENAR = 54568
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 41%
# PANJANG INPUT1 = 848
# PANJANG INPUT2 = 848
# PANJANG INPUT3 = 120
# BENAR = 54916
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 42%
# PANJANG INPUT1 = 4480
# PANJANG INPUT2 = 4480
# PANJANG INPUT3 = 4352
# BENAR = 57684
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 43%
# PANJANG INPUT1 = 2560
# PANJANG INPUT2 = 2560
# PANJANG INPUT3 = 56
# BENAR = 58658
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 44%
# PANJANG INPUT1 = 3456
# PANJANG INPUT2 = 3456
# PANJANG INPUT3 = 2496
# BENAR = 60578
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 45%
# PANJANG INPUT1 = 3328
# PANJANG INPUT2 = 3328
# PANJANG INPUT3 = 768
# BENAR = 62018
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 46%
# PANJANG INPUT1 = 704
# PANJANG INPUT2 = 704
# PANJANG INPUT3 = 240
# BENAR = 62342
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 47%
```

```
# PANJANG INPUT1 = 4480
# PANJANG INPUT2 = 4480
# PANJANG INPUT3 = 1728
# BENAR = 64454
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 48%
# PANJANG INPUT1 = 4032
# PANJANG INPUT2 = 4032
# PANJANG INPUT3 = 1600
# BENAR = 66366
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 49%
# PANJANG INPUT1 = 2880
# PANJANG INPUT2 = 2880
# PANJANG INPUT3 = 464
# BENAR = 67562
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 50%
# PANJANG INPUT1 = 440
# PANJANG INPUT2 = 440
# PANJANG INPUT3 = 296
# BENAR = 67801
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 51%
# PANJANG INPUT1 = 360
# PANJANG INPUT2 = 360
# PANJANG INPUT3 = 312
# BENAR = 68014
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 52%
# PANJANG INPUT1 = 320
# PANJANG INPUT2 = 320
# PANJANG INPUT3 = 200
# BENAR = 68184
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 53%
# PANJANG INPUT1 = 1088
# PANJANG INPUT2 = 1088
# PANJANG INPUT3 = 272
# BENAR = 68660
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
# LEVEL CONFIDENCE : 54%
# PANJANG INPUT1 = 1600
# PANJANG INPUT2 = 1600
# PANJANG INPUT3 = 184
# BENAR = 69306
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 55%
# PANJANG INPUT1 = 4672
# PANJANG INPUT2 = 4672
# PANJANG INPUT3 = 2176
# BENAR = 71602
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 56%
# PANJANG INPUT1 = 5888
# PANJANG INPUT2 = 5888
# PANJANG INPUT3 = 400
# BENAR = 73910
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 57%
# PANJANG INPUT1 = 5120
# PANJANG INPUT2 = 5120
# PANJANG INPUT3 = 3520
# BENAR = 76710
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 58%
# PANJANG INPUT1 = 5184
# PANJANG INPUT2 = 5184
# PANJANG INPUT3 = 304
# BENAR = 78730
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 59%
# PANJANG INPUT1 = 1280
# PANJANG INPUT2 = 1280
# PANJANG INPUT3 = 264
# BENAR = 79276
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 60%
# PANJANG INPUT1 = 1920
# PANJANG INPUT2 = 1920
# PANJANG INPUT3 = 944
# BENAR = 80232
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
```

```
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 61%
# PANJANG INPUT1 = 4096
# PANJANG INPUT2 = 4096
# PANJANG INPUT3 = 976
# BENAR = 82012
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 62%
# PANJANG INPUT1 = 1248
# PANJANG INPUT2 = 1248
# PANJANG INPUT3 = 408
# BENAR = 82582
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 63%
# PANJANG INPUT1 = 5632
# PANJANG INPUT2 = 5632
# PANJANG INPUT3 = 352
# BENAR = 84782
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 64%
# PANJANG INPUT1 = 2944
# PANJANG INPUT2 = 2944
# PANJANG INPUT3 = 384
# BENAR = 85982
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 65%
# PANJANG INPUT1 = 4544
# PANJANG INPUT2 = 4544
# PANJANG INPUT3 = 3328
# BENAR = 88518
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 66%
# PANJANG INPUT1 = 4096
# PANJANG INPUT2 = 4096
# PANJANG INPUT3 = 880
# BENAR = 90274
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 67%
# PANJANG INPUT1 = 408
# PANJANG INPUT2 = 408
# PANJANG INPUT3 = 168
# BENAR = 90469
# SALAH = 0
```

```
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 68%
# PANJANG INPUT1 = 6080
# PANJANG INPUT2 = 6080
# PANJANG INPUT3 = 928
# BENAR = 92981
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 69%
# PANJANG INPUT1 = 4224
# PANJANG INPUT2 = 4224
# PANJANG INPUT3 = 1984
# BENAR = 95061
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 70%
# PANJANG INPUT1 = 800
# PANJANG INPUT2 = 800
# PANJANG INPUT3 = 392
# BENAR = 95459
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 71%
# PANJANG INPUT1 = 1120
# PANJANG INPUT2 = 1120
# PANJANG INPUT3 = 136
# BENAR = 95913
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 72%
# PANJANG INPUT1 = 832
# PANJANG INPUT2 = 832
# PANJANG INPUT3 = 128
# BENAR = 96257
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 73%
# PANJANG INPUT1 = 3200
# PANJANG INPUT2 = 3200
# PANJANG INPUT3 = 1728
# BENAR = 97889
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 74%
# PANJANG INPUT1 = 5440
# PANJANG INPUT2 = 5440
# PANJANG INPUT3 = 200
# BENAR = 99979
```

```
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 75%
# PANJANG INPUT1 = 800
# PANJANG INPUT2 = 800
# PANJANG INPUT3 = 376
# BENAR = 100373
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 76%
# PANJANG INPUT1 = 4864
# PANJANG INPUT2 = 4864
# PANJANG INPUT3 = 1824
# BENAR = 102653
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 77%
# PANJANG INPUT1 = 3648
# PANJANG INPUT2 = 3648
# PANJANG INPUT3 = 2176
# BENAR = 104565
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 78%
# PANJANG INPUT1 = 5696
# PANJANG INPUT2 = 5696
# PANJANG INPUT3 = 272
# BENAR = 106769
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 79%
# PANJANG INPUT1 = 4800
# PANJANG INPUT2 = 4800
# PANJANG INPUT3 = 1056
# BENAR = 108833
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 80%
# PANJANG INPUT1 = 1216
# PANJANG INPUT2 = 1216
# PANJANG INPUT3 = 992
# BENAR = 109537
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 81%
# PANJANG INPUT1 = 488
# PANJANG INPUT2 = 488
# PANJANG INPUT3 = 248
```

```
# BENAR = 109782
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 82%
# PANJANG INPUT1 = 1664
# PANJANG INPUT2 = 1664
# PANJANG INPUT3 = 176
# BENAR = 110450
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 83%
# PANJANG INPUT1 = 4032
# PANJANG INPUT2 = 4032
# PANJANG INPUT3 = 592
# BENAR = 112110
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 84%
# PANJANG INPUT1 = 816
# PANJANG INPUT2 = 816
# PANJANG INPUT3 = 240
# BENAR = 112476
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 85%
# PANJANG INPUT1 = 1696
# PANJANG INPUT2 = 1696
# PANJANG INPUT3 = 1568
# BENAR = 113504
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 86%
# PANJANG INPUT1 = 3840
# PANJANG INPUT2 = 3840
# PANJANG INPUT3 = 1184
# BENAR = 115240
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 87%
# PANJANG INPUT1 = 1856
# PANJANG INPUT2 = 1856
# PANJANG INPUT3 = 1312
# BENAR = 116264
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 88%
# PANJANG INPUT1 = 1216
# PANJANG INPUT2 = 1216
```

```
# PANJANG INPUT3 = 960
# BENAR = 116960
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 89%
# PANJANG INPUT1 = 4096
# PANJANG INPUT2 = 4096
# PANJANG INPUT3 = 880
# BENAR = 118716
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 90%
# PANJANG INPUT1 = 312
# PANJANG INPUT2 = 312
# PANJANG INPUT3 = 72
# BENAR = 118851
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 91%
# PANJANG INPUT1 = 2016
# PANJANG INPUT2 = 2016
# PANJANG INPUT3 = 264
# BENAR = 119673
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 92%
# PANJANG INPUT1 = 3712
# PANJANG INPUT2 = 3712
# PANJANG INPUT3 = 1536
# BENAR = 121449
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 93%
# PANJANG INPUT1 = 4096
# PANJANG INPUT2 = 4096
# PANJANG INPUT3 = 640
# BENAR = 123145
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 94%
# PANJANG INPUT1 = 5760
# PANJANG INPUT2 = 5760
# PANJANG INPUT3 = 1152
# BENAR = 125593
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 95%
# PANJANG INPUT1 = 5376
```

```
# PANJANG INPUT2 = 5376
# PANJANG INPUT3 = 1312
# BENAR = 127937
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 96%
# PANJANG INPUT1 = 720
# PANJANG INPUT2 = 720
# PANJANG INPUT3 = 624
# BENAR = 128363
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 97%
# PANJANG INPUT1 = 1280
# PANJANG INPUT2 = 1280
# PANJANG INPUT3 = 800
# BENAR = 129043
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 98%
# PANJANG INPUT1 = 2112
# PANJANG INPUT2 = 2112
# PANJANG INPUT3 = 56
# BENAR = 129849
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 99%
# PANJANG INPUT1 = 1856
# PANJANG INPUT2 = 1856
# PANJANG INPUT3 = 256
# BENAR = 130609
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 100%
# PANJANG INPUT1 = 2240
# PANJANG INPUT2 = 2240
# PANJANG INPUT3 = 104
# BENAR = 131475
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
```

Hasil simulasi implementasi HW dan ABV turbo dekoder

```
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 2%
# PANJANG INPUT = 456
# BENAR = 56
# SALAH = 0
```

```
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 4%
# PANJANG INPUT = 184
# BENAR = 78
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 6%
# PANJANG INPUT = 896
# BENAR = 189
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 8%
# PANJANG INPUT = 1152
# BENAR = 332
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 10%
# PANJANG INPUT = 2016
# BENAR = 583
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 12%
# PANJANG INPUT = 576
# BENAR = 654
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 14%
# PANJANG INPUT = 1440
# BENAR = 833
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 16%
# PANJANG INPUT = 4480
# BENAR = 1392
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 18%
# PANJANG INPUT = 928
# BENAR = 1507
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 20%
# PANJANG INPUT = 360
# BENAR = 1551
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
```

```
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 22%
# PANJANG INPUT = 160
# BENAR = 1570
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 24%
# PANJANG INPUT = 1440
# BENAR = 1749
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 26%
# PANJANG INPUT = 48
# BENAR = 1755
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 28%
# PANJANG INPUT = 784
# BENAR = 1852
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 30%
# PANJANG INPUT = 2368
# BENAR = 2147
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 32%
# PANJANG INPUT = 3584
# BENAR = 2594
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 34%
# PANJANG INPUT = 1504
# BENAR = 2781
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 36%
# PANJANG INPUT = 1856
# BENAR = 3012
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 38%
# PANJANG INPUT = 3712
# BENAR = 3475
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
# LEVEL CONFIDENCE : 40%
# PANJANG INPUT = 288
# BENAR = 3510
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 42%
# PANJANG INPUT = 3264
# BENAR = 3917
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 44%
# PANJANG INPUT = 1632
# BENAR = 4120
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 46%
# PANJANG INPUT = 4032
# BENAR = 4623
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 48%
# PANJANG INPUT = 688
# BENAR = 4708
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 50%
# PANJANG INPUT = 800
# BENAR = 4807
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 52%
# PANJANG INPUT = 5824
# BENAR = 5534
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 54%
# PANJANG INPUT = 144
# BENAR = 5551
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 56%
# PANJANG INPUT = 408
# BENAR = 5601
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 58%
```

```
# PANJANG INPUT = 5184
# BENAR = 6248
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 60%
# PANJANG INPUT = 5120
# BENAR = 6887
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 62%
# PANJANG INPUT = 5632
# BENAR = 7590
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 64%
# PANJANG INPUT = 672
# BENAR = 7673
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 66%
# PANJANG INPUT = 672
# BENAR = 7756
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 68%
# PANJANG INPUT = 624
# BENAR = 7833
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 70%
# PANJANG INPUT = 1376
# BENAR = 8004
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 72%
# PANJANG INPUT = 120
# BENAR = 8018
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 74%
# PANJANG INPUT = 4608
# BENAR = 8593
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 76%
# PANJANG INPUT = 2112
```

```
# BENAR = 8856
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 78%
# PANJANG INPUT = 320
# BENAR = 8895
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 80%
# PANJANG INPUT = 2688
# BENAR = 9230
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 82%
# PANJANG INPUT = 848
# BENAR = 9335
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 84%
# PANJANG INPUT = 4480
# BENAR = 9894
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 86%
# PANJANG INPUT = 56
# BENAR = 9901
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 88%
# PANJANG INPUT = 2496
# BENAR = 10212
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 90%
# PANJANG INPUT = 768
# BENAR = 10307
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 92%
# PANJANG INPUT = 704
# BENAR = 10394
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 94%
# PANJANG INPUT = 1728
# BENAR = 10609
```

```
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 96%
# PANJANG INPUT = 4032
# BENAR = 11112
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 98%
# PANJANG INPUT = 2880
# BENAR = 11471
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# LEVEL CONFIDENCE : 100%
# PANJANG INPUT = 440
# BENAR = 11525
# SALAH = 0
# vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
```