APLIKASI PENGANALISIS SPEKTRUM SINYAL DIGITAL SECARA *REAL-TIME* MENGGUNAKAN ALGORITMA *FAST FOURIER TRANSFORM* (FFT) PADA PERANGKAT *MOBILE PHONE*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh:
ASRI SAMSIAR ILMANANDA
NIM: 0710630027 - 63

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG
2012

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

LEMBAR PERSETUJUAN

APLIKASI PENGANALISIS SPEKTRUM SINYAL DIGITAL SECARA *REAL-TIME* MENGGUNAKAN ALGORITMA *FAST FOURIER TRANSFORM* (FFT) PADA PERANGKAT *MOBILE PHONE*

SKRIPSI
JURUSAN TEKNIK ELEKTRO

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh :

ASRI SAMSIAR ILMANANDA
NIM: 0710630027 – 63

Telah diperiksa dan disetujui oleh

Dosen Pembimbing

Pembimbing I	**Pembimbing II**
Muhammad Aswin, Ir., MT.	**Ali Mustofa, ST., MT.**
NIP. 19640626 199002 1 001	**NIP. 19710601 200003 1 001**

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

LEMBAR PENGESAHAN

APLIKASI PENGANALISIS SPEKTRUM SINYAL DIGITAL SECARA *REAL-TIME* MENGGUNAKAN ALGORITMA *FAST FOURIER TRANSFORM* (FFT) PADA PERANGKAT *MOBILE PHONE*

Disusun Oleh :

ASRI SAMSIAR ILMANANDA
NIM: 0710630027 – 63

Skripsi ini telah diuji dan dinyatakan lulus
pada tanggal **31 Januari 2012**

Dosen Penguji :

Waru Djuriatno. ST., MT.	**R. Arief Setyawan, ST., MT.**
NIP. 19690725 199702 1 001	**NIP. 19750819 199903 1 001**

Adharul Muttaqin, ST., MT.
NIP. 19760121 200501 1 001

Mengetahui :
Ketua Jurusan Teknik Elektro

Dr. Ir. Sholeh Hadi Pramono ,MS.
NIP. 19580728 198701 1 001

UNIVERSITAS BRAWIJAYA

KATA PENGANTAR

Dengan mengucapkan Alhamdulillahi Rabbil'alamin, puji syukur kepada Allah SWT yang telah melimpahkan rahmat, ridho, taufik dan hidayah-Nya sehingga penulis dapat menyelesaikan skripsi dengan judul "Aplikasi Penganalisis Spektrum Sinyal Digital secara *Real-Time* Menggunakan Algoritma *Fast Fourier Transform* (FFT) pada Perangkat *Mobile Phone*" sebagai salah satu syarat yang diajukan untuk memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya Malang.

Penulis menyadari bahwa penulisan skripsi ini tidak akan terselesaikan tanpa adanya bantuan dari berbagai pihak karena itu ucapan terima kasih penulis sampaikan kepada semua pihak yang telah dengan rela meluangkan waktunya untuk membantu penulis dalam proses penulisan skripsi dan juga dengan ikhlas memberikan bimbingan, arahan, masukan, dan semangat sehingga penulis dapat menyelesaikan skripsi ini. Oleh karena itu pada kesempatan ini, penulis mengucapkan terima kasih kepada:

1. Bapak Dr. Ir. Sholeh Hadi Pramono, MS. selaku Ketua Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
2. Bapak M. Azis Muslim, S.T., M.T., PhD. selaku Sekretaris Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
3. Bapak Moch. Rif'an, ST., MT. selaku Ketua Program Studi Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
4. Bapak Waru Djuriatno, ST., MT. selaku Ketua Kelompok Dosen Keahlian Rekayasa Komputer.
5. Bapak Muhammad Aswin, Ir., MT. serta Bapak Ali Mustofa, ST., MT. selaku dosen pembimbing I dan dosen pembimbing II yang telah banyak memberikan bimbingan, masukan dan arahan dalam penyusunan skripsi ini.
6. Bapak Hadi Suyono, ST., MT., PhD. selaku dosen penasehat akademik yang telah memberikan nasehat dan arahan dalam proses akademik penulis.
7. Ayah Yunan Rusdianto, Ibu S. Chamidah Sri H., Kakak Anisa Imaniar E. P. serta Adik Rizki Amalia T.C. atas segala dukungan dan do'a serta dorongan semangat hingga terselesaikannya skripsi ini.
8. Ferdiansyah Gustiawan beserta keluarga yang selalu memberi dukungan dan semangat kepada penulis selama proses pengerjaan skripsi ini.

9. Rekan dalam pengerjaan skripsi Nourma Devita Pratiwi, Andik Nur Achmad, Ade Prima K. W, Fanni Kharisma yang telah membantu memberi solusi pada permasalahan yang muncul dan memberi masukan dalam proses pengerjaan skripsi ini.
10. Yayuk I., Inas W. H., FarizzaA. beserta sahabat-sahabat CoreNerz Girls yang telah menemani dalam suka maupun duka dalam kebersamaan yang penuh canda tawa.
11. Seluruh teman-teman angkatan 2007 (CORE '07) yang selalu bersama dalam keceriaan.
12. Rekan-rekan Laboratorium Dasar Elektrik dan Pengukuran.
13. Semua pihak yang telah membantu penyelesaian skripsi ini baik secara langsung dan tidak langsung yang tidak dapat penulis sebutkan satu per satu.

Penulis berharap semoga skripsi ini bermanfaat bagi pembaca dan dapat memberikan sumbangan pikiran bagi pihak-pihak lain khususnya mahasiswa jurusan Teknik Elektro. Segala kritik dan saran sangat penulis harapkan demi kesempurnaan skripsi ini, karena penulis menyadari bahwa penulisan skripsi ini masih jauh dari sempurna dan masih memiliki banyak kekurangan.

Malang, Januari 2012

Penulis

DAFTAR ISI

KATA PENGANTAR ..... **i**
DAFTAR ISI ..... **iii**
DAFTAR GAMBAR ..... **vii**
DAFTAR TABEL ..... **x**
ABSTRAK ..... **xi**
BAB I **1**
PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 1
1.3 Batasan Masalah 2
1.4 Tujuan 2
1.5 Manfaat 2
1.6 Sistematika Penulisan 3
BAB II **4**
TINJAUAN PUSTAKA **4**
2.1 Audio 4
2.1.1 Bentuk Dasar Gelombang Audio 4
2.1.2 Karakteristik Sinyal Audio 6
2.1.3 Frekuensi, Periode dan Fasa 7
2.2 Desibel 7
2.2.1 Hubungan dengan Daya 8
2.2.2 Hubungan Tegangan, Arus dan Tekanan 8
2.2.3 *Sound Pressure Level* (SPL) 9
2.3 Sinyal dan Pemrosesan Sinyal Digital 10
2.3.1 Pengertian Dasar Sinyal 10
2.3.2 Pemrosesan Sinyal Digital 12
2.4 Pemrosesan Audio Digital 14
2.4.1 *Pulse Code Modulation* (PCM) 14
2.4.2 *Audio Quality* 15
2.4.3 Transduser 15
2.4.3.1 *Microphone* 16
2.4.3.1.1 Prinsip Kerja *Microphone* 16

2.4.3.1.2 *Microphone* Kondenser 16
2.4.3.1.3 *Microphone* Kondenser Elektret 17
2.4.3.1.4 Respon Frekuensi *Microphone* 17
2.4.3.2 *Loudspeaker* 18
2.4.4 *Recording* dan *Playback* 20
2.4.5 Pembentukan Data Digital 22
2.5 *Spectrum Analyzer* 22
2.6 *Fast Fourier Transform* (FFT) 24
2.6.1 *Discrete Fourier Transform* (DFT) 25
2.6.2 Pembentukan algoritma FFT *Decimation In Time* 28
2.6.3 *Bit Reversed* 29
2.6.4 *Twiddle Factor* 30
2.6.5 Diagram *Butterfly* 31
2.6.6 Perbandingan Komputasi DFT dan FFT 32
2.7 Kompleksitas Algoritma dan Notasi *Big-O* 33
2.8 *Windows Mobile* 34
2.8.1 *Waveform Audio* 35
2.8.2 *Graphics Device Interface* (GDI) 35
2.9 Bahasa Pemrograman C++ 35
BAB III 37
METODOLOGI PENELITIAN 37
3.1 Studi Literatur 37
3.2 Perancangan dan Implementasi Sistem 37
3.3 Pengujian Sistem 39
3.4 Pengambilan Kesimpulan dan Saran 40
BAB IV 41
PERANCANGAN DAN IMPLEMENTASI 41
4.1 Perancangan Secara Umum 41
4.1.1 Blok Diagram Sistem 41
4.1.2 Cara Kerja Aplikasi 43
4.1.3 Konfigurasi Perangkat Keras 44
4.2 Perancangan Perangkat Lunak 44
4.2.1 Perekaman Suara (*Recording*) 45
4.2.2 Mekanisme Penyimpanan dan Pemrosesan Data 47

4.2.3 Algoritma FFT 50
4.2.4 Pembaharuan (*Update*) Tampilan pada Layar 57
4.2.4.1 Tampilan Mode Grafik Sinusoida 58
4.2.4.2 Tampilan Mode Grafik Spektrum Frekuensi Skala Linear 59
4.2.4.3 Tampilan Mode Grafik Spektrum Frekuensi Skala Logaritmik 61
4.2.5 Perancangan Aplikasi Keseluruhan 62
4.2.6 Menghentikan (*Stop*) Aplikasi 64
4.3 Implementasi Sistem 66
4.3.1 Lingkungan Implementasi 66
4.3.2 Implementasi Perekaman Suara (*Recording*) 67
4.3.3 Implementasi Algoritma FFT 70
4.3.4 Implementasi Pembaharuan (*Update*) Tampilan pada Layar 72
4.3.4 Implementasi Menghentikan Aplikasi (*Stop*) 76
4.4 Implementasi Antarmuka (*Interface*) 77
4.4.1 Implementasi Antarmuka Menu Pengaturan 78
4.4.2 Implementasi Antarmuka Tampilan Grafik 79
4.5 Pembuatan Paket Instalasi 80
BAB V 81
PENGUJIAN 81
5.1 Pengujian Hasil Tampilan 81
5.1.1 Hasil Tampilan Grafik Sinyal pada Kondisi 1 (Volume 0/nol) 83
5.1.2 Hasil Tampilan Grafik Sinyal pada Kondisi 2 (Volume ±25% skala) 84
5.1.3 Hasil Tampilan Grafik Sinyal pada Kondisi 3 (Volume maksimum) 86
5.1.4 Hasil Tampilan Grafik pada Kondisi 4 (Dua sinyal *input* yang diputar bersamaan) 88
5.2 Kompleksitas Algoritma 90
5.2.1 Kompleksitas Algoritma Grafik Sinusoida 91
5.2.2 Kompleksitas Algoritma Grafik Spektrum Frekuensi Skala Linear 91
5.2.3 Kompleksitas Algoritma Grafik Spektrum Frekuensi Skala Logaritmik 93
5.3 Analisis Sistem 94
BAB VI 95
PENUTUP 95
6.1 Kesimpulan 95
6.2 Saran 96

DAFTAR PUSTAKA..... 97

DAFTAR GAMBAR

Gambar 2.1 Sinyal audio 4
Gambar 2.2 Gelombang pada nada konstan frekuensi tertentu 5
Gambar 2.3 Hubungan bentuk gelombang dengan kerapatan molekul udara 5
Gambar 2.4 Panjang Gelombang 6
Gambar 2.5 Amplitudo 6
Gambar 2.6 Frekuensi 7
Gambar 2.7 Tingkat tekanan suara 10
Gambar 2.8 Sinyal analog 11
Gambar 2.9 Sinyal digital 11
Gambar 2.10 Langkah pemrosesan sinyal analog menjadi sinyal digital 13
Gambar 2.11 Potongan melintang *microphone* kondenser 17
Gambar 2.12 *Microphone* kondenser elektret 17
Gambar 2.13 Respon frekuensi *microphone* kondenser elektret secara umum 18
Gambar 2.14 *Loudspeaker* 18
Gambar 2.15 Penampang *loudspeaker* 18
Gambar 2.16 Respon frekuensi *loudspeaker* secara umum (*high-quality*) 20
Gambar 2.17 Diagram pemrosesan audio digital 21
Gambar 2.18 Diagram hubungan domain waktu dengan domain frekuensi 23
Gambar 2.19 Format dasar tampilan *spectrum analyzer* 23
Gambar 2.20 Ilustrasi Transformasi *Fourier* untuk konversi sinyal 24
Gambar 2.21 Sinyal fungsi waktu (sebelum perhitungan DFT) 25
Gambar 2.22 Spektrum frekuensi dari sinyal fungsi waktu (setelah perhitungan DFT) 27
Gambar 2.23 Ilustrasi penguraian DFT 29
Gambar 2.24 Lingkaran *twiddle factor* untuk N = 8 30
Gambar 2.25 Diagram dasar *butterfly* 31
Gambar 2.26 Diagram *butterfly* untuk *4-Point* FFT 32
Gambar 2.27 Fitur-fitur pada *Windows Mobile* 34
Gambar 3.1 Desain umum rancangan sistem 38
Gambar 4.1 Diagram blok sistem 41
Gambar 4.2 Blok diagram perubahan bentuk sinyal *input* untuk menampilkan spektrum frekuensi 42

Gambar 4.3 Sebaran suara yang dapat memperbesar *noise* sinyal *input* 44
Gambar 4.4 Diagram alir persiapan perekaman suara 46
Gambar 4.5 Diagram alir perekaman suara..... 47
Gambar 4.6 Mekanisme penyimpanan data pada *buffer*..... 48
Gambar 4.7 Ilustrasi hubungan antara penyimpanan data dengan pemrosesan data 49
Gambar 4.8 Diagram alir persiapan perhitungan FFT 50
Gambar 4.9 Diagram *butterfly* untuk *256-Point* FFT 53
Gambar 4.10 Diagram alir algoritma FFT 55
Gambar 4.11 Tampilan layar secara umum 57
Gambar 4.12 Diagram alir tampilan mode grafik sinusoida 58
Gambar 4.13 Diagram alir tampilan mode grafik spektrum frekuensi skala linear 60
Gambar 4.14 Diagram alir tampilan mode grafik spektrum frekuensi skala logaritmik 62
Gambar 4.15 Diagram alir perancangan aplikasi keseluruhan..... 63
Gambar 4.16 Diagram alir menghentikan aplikasi 65
Gambar 4.17 Tampilan utama aplikasi penganalisis spektrum sinyal digital 78
Gambar 4.18 Tampilan menu pengaturan..... 78
Gambar 4.19 Tampilan grafik sinusoida..... 79
Gambar 4.20 Tampilan grafik spektrum frekuensi skala linear..... 79
Gambar 4.21 Tampilan grafik spektrum frekuensi skala logaritmik 80
Gambar 5.1 Hasil instalasi aplikasi..... 81
Gambar 5.2 Persiapan pengujian aplikasi 82
Gambar 5.3 Tampilan grafik sinusoida sinyal pada kondisi 1 83
Gambar 5.4 Tampilan grafik spektrum frekuensi (linear) sinyal pada kondisi 1... 83
Gambar 5.5 Tampilan grafik spektrum frekuensi (logaritmik) sinyal pada kondisi 1 84
Gambar 5.6 Tampilan grafik sinusoida sinyal pada kondisi 2 85
Gambar 5.7 Tampilan grafik spektrum frekuensi (linear) sinyal pada kondisi 2... 85
Gambar 5.8 Tampilan grafik spektrum frekuensi (logaritmik) sinyal pada kondisi 2 86
Gambar 5.9 Tampilan grafik sinusoida sinyal pada kondisi 3 87
Gambar 5.10 Tampilan grafik spektrum frekuensi (linear) sinyal pada kondisi 3... 87

Gambar 5.11 Tampilan grafik spektrum frekuensi (logaritmik) sinyal pada kondisi 3 88
Gambar 5.12 Tampilan grafik sinusoida pada kondisi 4..... 89
Gambar 5.13 Tampilan grafik spektrum frekuensi (linear) pada kondisi 4 89
Gambar 5.14 Tampilan grafik spektrum frekuensi (logaritmik) pada kondisi 4..... 90
Gambar 5.15 Ilustrasi komputasi FFT 92

DAFTAR TABEL

Tabel 2.1 Perhitungan desibel dengan nilai P_o = 1 watt 8

Tabel 2.2 Audio quality yang digunakan pada digital recording 15

Tabel 2.3 Level kuantisasi untuk *bit depth* = 3 bit 22

Tabel 2.4 Susunan *bit-reversed* data *input* untuk N = 8 30

Tabel 4.1 Perhitungan *bit-reversed* untuk nilai i = 00010110 51

Tabel 4.2 Hasil perhitungan *bit-reversed* 256 titik 52

Tabel 4.3 Parameter-parameter perhitungan *256-Point* FFT 54

Tabel 4.4 Struct dan function pada Waveform Audio API 67

Tabel 4.5 *Struct* dan *function* pada GDI 73

repository.ub.ac.id

ABSTRAK

ASRI SAMSIAR ILMANANDA, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Januari, 2012. *Aplikasi Penganalisis Spektrum Sinyal Digital Secara Real-Time Menggunakan Algoritma Fast Fourier Transform (FFT) Pada Perangkat Mobile Phone*, Dosen Pembimbing: Muhammad Aswin, Ir., MT. dan Ali Mustofa, ST., MT.

Perangkat penganalisis spektrum atau biasa disebut *spectrum analyzer* adalah suatu perangkat yang dapat digunakan untuk mengetahui komposisi spektrum dari suatu sinyal. Salah satu metode yang digunakan pada analisis spektrum adalah *Fast Fourier Transform* (FFT), yang melakukan pengolahan sinyal secara digital. Teknologi terkomputerisasi untuk aplikasi-aplikasi pemrosesan sinyal digital telah mulai dikembangkan, khususnya analisis spektrum sinyal audio pada perangkat *mobile phone*. Akan tetapi analisis spektrum pada perangkat *mobile phone* umumnya hanya dilakukan pada *file-file* suara saja, tidak dilakukan secara *real-time* dari sinyal asal yaitu suara. Padahal, analisis spektrum secara *real-time* dapat digunakan untuk menggantikan fungsi perangkat *spectrum analyzer*, yaitu untuk menganalisis kualitas sinyal melalui pengamatan spektrum frekuensi. Hal ini sangat bermanfaat untuk media pembelajaran awal dan dapat dikembangkan lebih lanjut, khususnya pada perangkat *mobile phone* yang memiliki keterbatasan dalam hal pemrosesan sinyal. Oleh karena itu, dibuatlah sebuah aplikasi penganalisis spektrum sinyal digital pada perangkat *mobile phone* sebagai alternatif untuk analisis spektrum secara *real-time*.

Skripsi ini membahas tentang bagaimana cara merancang dan membangun aplikasi penganalisis spektrum sinyal digital pada perangkat *mobile phone* dengan menggunakan metode FFT. Aplikasi ini memiliki dua proses utama yang berjalan terpisah, yaitu perekaman suara (*recording*) dan pemrosesan data. Kedua proses ini berjalan secara hampir bersamaan, atau disebut sebagai *parallel processing*. Pemrosesan data dilakukan secara kontinyu dan cepat, sehingga dapat dihasilkan sebuah sistem aplikasi yang bekerja dengan prinsip waktu-nyata (*real-time*). Aplikasi ini memiliki tiga macam pilihan tampilan sinyal yaitu grafik sinusoida, grafik spektrum frekuensi skala linear dan grafik spektrum frekuensi skala logaritmik. Untuk menampilkan sinyal dalam *domain* frekuensi, data akan diproses telebih dahulu menggunakan algoritma FFT.

Hasil dari aplikasi ini adalah tampilan sinyal dalam *domain* waktu maupun *domain* frekuensi yang merupakan representasi dari sinyal masukan. Hasil tampilan

UNIVERSITAS BRAWIJAYA

dipengaruhi oleh tingkat kenyaringan suara (*sound pressure level*). Kompleksitas algoritma untuk menampilkan grafik sinusoida adalah O(*N)*. Sedangkan kompleksitas algoritma untuk menampikan grafik spektrum frekuensi skala linear maupun skala logaritmik adalah $O(N\log_2 N)$.

Kata kunci: *Spectrum analyzer, audio, mobile phone, Fast Fourier Transform (FFT).*

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perangkat penganalisis spektrum atau biasa disebut *spectrum analyzer* adalah suatu perangkat yang dapat digunakan untuk mengetahui komposisi spektrum dari suatu sinyal. Jika pada umumnya sinyal ditampilkan dalam kawasan waktu, *spectrum analyzer* menampilkan sinyal dalam kawasan frekuensi. Salah satu metode yang digunakan pada analisis spektrum adalah *Fast Fourier Transform* (FFT).

FFT Spectrum Analyzer bekerja dengan mencuplik sinyal informasi (*sampling*) yang hasilnya akan diolah secara digital. Dengan prinsip yang sama, mulai dikembangkan teknologi terkomputerisasi untuk aplikasi-aplikasi pemrosesan sinyal digital, termasuk aplikasi penganalisis spektrum sinyal audio pada perangkat *mobile phone*.

Pada umumya analisis spektrum pada perangkat *mobile phone* hanya dilakukan pada *file-file* suara saja, yang berarti data olahan sudah berupa data digital. Aplikasi ini tidak melakukan proses konversi secara langsung dari sinyal asal yaitu suara (*real-time*). Padahal, analisis spektrum secara *real-time* dapat digunakan untuk menggantikan fungsi perangkat *spectrum analyzer*, yaitu untuk menganalisis kualitas sinyal melalui pengamatan spektrum frekuensi. Hal ini sangat bermanfaat untuk media pembelajaran awal dan dapat dikembangkan lebih lanjut, khususnya pada perangkat *mobile phone* yang memiliki keterbatasan dalam hal pemrosesan sinyal.

Oleh karena itu, dibuatlah sebuah aplikasi penganalisis spektrum sinyal digital pada perangkat *mobile phone* sebagai alternatif untuk analisis sinyal secara *real-time*. Aplikasi ini melakukan pemrosesan pada sinyal audio (suara) dan menggunakan metode FFT.

1.2 Rumusan Masalah

Adapun rumusan masalah dalam skripsi ini dengan mengacu pada uraian latar belakang adalah sebagai berikut:

1. Bagaimana cara memproses data masukan berupa suara?
2. Bagaimana algoritma perhitungan FFT yang digunakan?
3. Bagaimana implementasi dari algorima tersebut agar dapat menampilkan spektrum frekuensi sinyal secara *real-time* pada perangkat *mobile phone*?

1.3 Batasan Masalah

Pembahasan dalam skripsi ini dilakukan dengan batasan sebagai berikut:

1. Sinyal masukan berupa sinyal audio (suara),
2. Media untuk sinyal masukan adalah *microphone* (*mobile phone*),
3. Resolusi layar *mobile phone* yang digunakan yaitu 320 x 240 pixel,
4. Format *recording* untuk pengambilan *sample* yaitu *mono*, 8 bit, 8 kHz,
5. Algoritma yang digunakan adalah *Fast Fourier Transform Decimation in Time*,
6. Aplikasi dikembangkan pada perangkat berbasis sistem operasi *Windows Mobile* menggunakan bahasa pemrograman C++.

1.4 Tujuan

Skripsi ini bertujuan untuk merancang dan membangun aplikasi penganalisis spektrum sinyal digital pada perangkat *mobile phone* dengan menggunakan metode FFT. Aplikasi ini dapat menampilkan spektrum frekuensi sinyal secara *real-time* dari sinyal masukan berupa suara.

1.5 Manfaat

Manfaat yang dapat diperoleh dengan penulisan skripsi ini diantaranya adalah sebagai sebagai berikut:

a) Bagi penyusun:
   1. Mampu membangun aplikasi penganalisis spektrum sinyal digital yang dijalankan pada perangkat *mobile phone*.
   2. Menambah wawasan mengenai pemrosesan dan analisis spektrum sinyal digital pada perangkat *mobile phone*.
   3. Menerapkan ilmu yang telah diperoleh dalam perkuliahan.

b) Bagi pengguna:
   1. Dapat memanfaatkan perangkat *mobile phone* menjadi sebuah perangkat penganalisis spektrum.
   2. Memberikan alternatif untuk analisis spektrum secara praktis dan dengan harga yang terjangkau.

1.6 Sistematika Penulisan

Skripsi ini disusun dalam enam bab dengan sistematika penulisan sebagai berikut:

BAB I PENDAHULUAN

Memuat latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Mengkaji teori-teori yang menunjang dalam perancangan dan pembuatan perangkat lunak.

BAB III METODOLOGI PENELITIAN

Membahas metode-metode yang digunakan dalam perancangan dan pembuatan perangkat lunak.

BAB IV PERANCANGAN DAN IMPLEMENTASI

Menjelaskan langkah-langkah perancangan yang digunakan dalam pembuatan perangkat lunak, serta membahas implementasi dari perancangan perangkat lunak ke dalam algoritma pemrograman.

BAB V PENGUJIAN

Melakukan pengujian terhadap perangkat lunak serta menjelaskan analisis dari hasil pengujian.

BAB VI PENUTUP

Memuat kesimpulan yang diperoleh serta saran untuk pengembangan dan penyempurnaan skripsi ini.

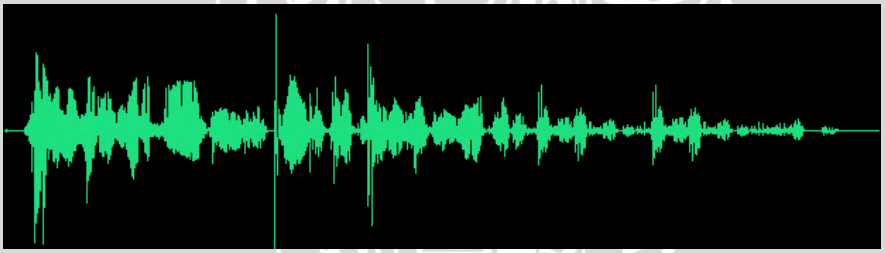
BAB II
TINJAUAN PUSTAKA

2.1 Audio

Audio diartikan sebagai suara atau reproduksi suara. Gelombang suara merupakan gelombang longitudinal yang dihasilkan dari sebuah benda yang bergetar. Getaran benda merambat melalui medium atau zat perantara yang dapat berupa zat cair, padat dan gas (udara). Suara tidak dapat merambat melalui ruangan hampa udara.

Gelombang suara memiliki lembah dan bukit, satu buah lembah dan bukit akan menghasilkan satu siklus atau periode. Siklus ini berlangsung berulang-ulang, yang membawa pada konsep frekuensi. Frekuensi adalah jumlah dari siklus yang terjadi dalam satu detik. Satuan dari frekuensi adalah Hertz atau disingkat Hz.

Sinyal audio adalah sinyal yang dapat diterima oleh telinga manusia. Telinga manusia dapat mendengar bunyi antara 20 Hz hingga 20 kHz (20.000 Hz). Angka 20 Hz sebagai frekuensi suara terendah yang dapat didengar, sedangkan 20 kHz merupakan frekuensi tertinggi yang dapat didengar.

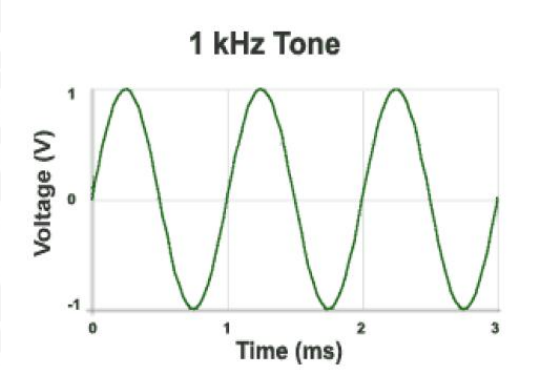


Gambar 2.1 Sinyal audio
(**Sumber :** http://www.niu.edu)

2.1.1 Bentuk Dasar Gelombang Audio

Gelombang suara bervariasi sebagaimana variasi tekanan media perantara seperti udara. Suara diciptakan oleh getaran dari suatu obyek, yang menyebabkan udara disekitarnya bergetar. Getaran udara ini kemudian menyebabkan gendang telinga manusia bergetar, yang oleh otak di interpretasikan sebagai suara.

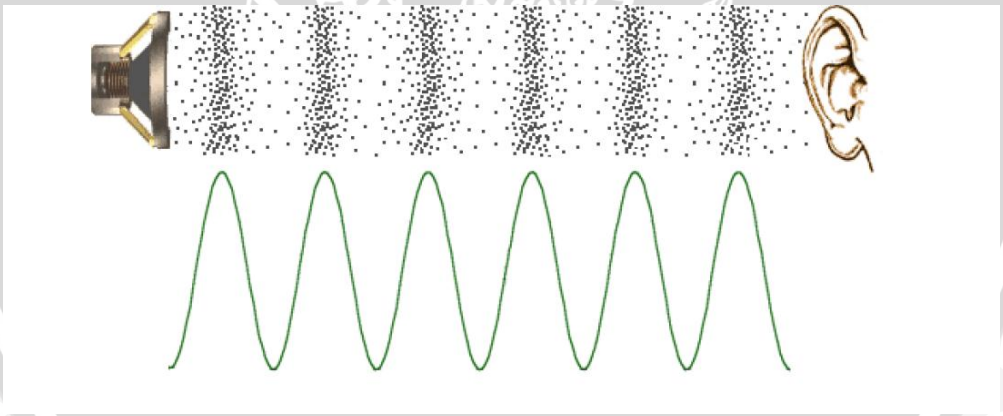
Ilustrasi gelombang suara yang merambat melalui udara dapat dianalogikan dengan perjalanan gelombang air melalui air. Gelombang suara dapat juga ditunjukkan dalam suatu grafik standar x terhadap y seperti ditunjukkan gambar 2.2.



Gambar 2.2 Gelombang pada nada konstan frekuensi tertentu
(**Sumber :** Sri Waluyanti dkk, 2008)

Gambar 2.2 merupakan visualisasi gelombang dengan sudut pandang matematis. Gelombang ditunjukkan pada nada konstan frekuensi tertentu, dan menghasilkan kurva yang dikenal sebagai bentuk gelombang.

Bentuk gelombang berkaitan dengan kerapatan molekul udara. Bila bentuk gelombang pada titik tinggi, maka molekul udara menyebar rapat. Bila gelombang pada titik rendah, maka molekul udara menyebar lebih tipis. Hubungan bentuk gelombang dengan kerapat molekul udara ditunjukkan pada Gambar 2.3.



Gambar 2.3 Hubungan bentuk gelombang dengan kerapatan molekul udara
(**Sumber :** Sri Waluyanti dkk, 2008)

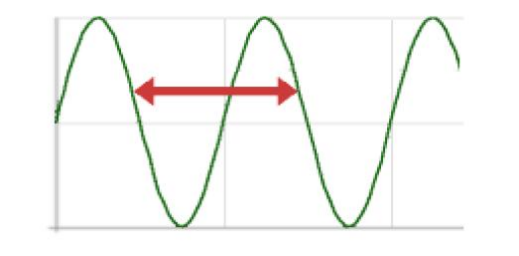
Titik hitam menunjukkan molekul udara. Getaran *loudspeaker* menyebabkan molekul udara disekitarnya bergetar dalam pola tertentu yang ditunjukkan dengan bentuk gelombang. Getaran udara ini menyebabkan gendang telinga pendengar bergetar dengan pola yang sama. Molekul udara pada dasarnya tidak berjalan dari *loudspeaker* ke telinga. Setiap molekul udara yang berpindah pada jarak kecil sebagai getaran,

sebenarnya dikarenakan molekul yang bersebelahan terpengaruh sehingga bergetar dan berjalan sampai telinga.

2.1.2 Karakteristik Sinyal Audio

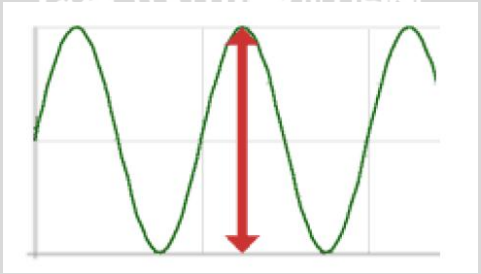
Sinyal audio dapat terbentuk dari perpaduan sejumlah gelombang sinusoida dengan karakteristik yang berbeda-beda. Tiga karakteristik dasar sinyal audio antara lain: panjang gelombang, amplitudo dan frekuensi.

Panjang gelombang adalah jarak antar titik gelombang dan titik ekuivalen pada siklus/periode berikutnya. Panjang gelombang dinyatakan dalam satuan meter. Panjang Gelombang ditunjukkan pada Gambar 2.4.



Gambar 2.4 Panjang Gelombang
(**Sumber :** Sri Waluyanti dkk, 2008)

Amplitudo adalah tinggi gelombang yang berupa kekuatan atau daya gelombang sinyal. Amplitudo ditunjukkan pada Gambar 2.5. Gelombang yang lebih tinggi diinterpretasikan sebagai volume yang lebih tinggi.



Gambar 2.5 Amplitudo
(**Sumber :** Sri Waluyanti dkk, 2008)

Frekuensi adalah jumlah getaran yang terjadi dalam waktu satu detik. Frekuensi diukur dalam hertz atau siklus per detik. Semakin cepat getaran gelombang suara, maka frekuensi semakin tinggi. Frekuensi lebih tinggi diinterpretasikan sebagai jalur yang lebih tinggi (semakin nyaring).

Gambar 2.6 Frekuensi
(**Sumber :** Sri Waluyanti dkk, 2008)

2.1.3 Frekuensi, Periode dan Fasa

Frekuensi adalah jumlah getaran yang terjadi dalam waktu satu detik atau banyaknya gelombang/getaran listrik yang dihasilkan tiap detik. Frekuensi dilambangkan dalam huruf *f*. Sedangkan periode adalah selang waktu yang diperlukan untuk melakukan satu getaran sempurna. Periode dilambangkan dengan huruf *T*. Hubungan antara frekuensi dan periode adalah berbanding terbalik, artinya semakin besar frekuensi, maka periode akan semakin kecil. Secara matematis dapat dituliskan sebagai berikut.

$$f = \frac{1}{T} \quad ; \quad T = \frac{1}{f} \tag{2-1}$$

dimana:

f : frekuensi, dalam siklus per detik atau Hertz

T : periode, dalam detik.

Fasa adalah pergeseran periode waktu dari posisi baris nol. Fasa dinyatakan sebagai besar sudut pada waktu tertentu. Pada beda fasa dikenal istilah "*lag*" atau "*lead*". *Lag* artinya harga maksimum atau nol yang dicapai satu siklus lebih lambat atau ketinggalan dari siklus lainnya. Sedangkan *lead* artinya harga maksimum atau nol yamg dicapai satu siklus mendahului siklus lainnya.

2.2 Desibel

Desibel digunakan dalam teknologi audio untuk mengekpresikan tingkatan sinyal dan perbedaan tekanan suara, daya, tegangan dan arus. Desibel berguna untuk mengukur perbandingan dalam cakupan angka-angka kecil untuk menyatakan suatu besaran agar dapat dianalisis dengan lebih mudah.

2.2.1 Hubungan dengan Daya

Pada dasarnya bel didefinisikan sebagai perbandingan daya (P) terhadap referensinya secara logaritmis. Satuan yang lebih umum digunakan adalah desibel yaitu sepersepuluh dari bel.

$$Bell = \log(\frac{P_1}{P_o}) \quad (2\text{-}2)$$

$$decibel\ (dB) = 10\log(\frac{P_1}{P_o}) \quad (2\text{-}3)$$

Contoh perhitungan desibel dengan nilai P_o = 1 watt adalah sebagai berikut.

Tabel 2.1 Perhitungan desibel dengan nilai P_o = 1 watt

P_1 (watt)	Tingkatan dalam dB	P_1 (watt)	Tingkatan dalam dB
1	0	1000	30
10	10	10000	40
100	20	20000	43

(**Sumber :** Sri Waluyanti dkk, 2008)

Dalam kawasan teknologi audio, penambahan daya sepuluh kali akan menghasilkan tingkatan yang pada umumnya dapat dikatakan dua kali lebih nyaring. Sinyal akustik 100 watt menjadi dua kali lebih keras dari sinyal 10 watt dan sinyal 10 watt akan dirasa dua kali lebih keras dari pada sinyal 1 watt. Masing-masing perbandingan daya dapat diekspresikan dengan level yang sama 10 dB.

2.2.2 Hubungan Tegangan, Arus dan Tekanan

Desibel pada dasarnya merupakan hubungan perbandingan daya (P) dan dapat diterapkan pada tegangan (E), arus (I) dan tekanan sebagai hubungan daya. Daya listrik dapat direpresentasikan dalam persamaan sebagai berikut.

$$P = I^2Z = E^2/Z \quad (2\text{-}3)$$

Karena daya berupa perbandingan kuadrat dari tegangan maupun arus, maka tingkat daya dalam dB dapat dihitung menggunakan persamaan berikut.

$$decibel\ (dB) = 10\log\left(\frac{\left(\frac{E_1}{Z}\right)^2}{\left(\frac{E_o}{Z}\right)^2}\right)$$

$$= 10\log(\frac{E_1}{E_o})^2 = 20\log(\frac{E_1}{E_o}) = 20\log(\frac{I_1}{I_o}) \quad (2\text{-}4)$$

Tingkat dB dari tekanan suara sebanding dengan tingkat dB dari arus maupun tegangan.

$$decibel\ (dB) = 20 \log \left(\frac{P_1}{P_o}\right) \quad (2\text{-}5)$$

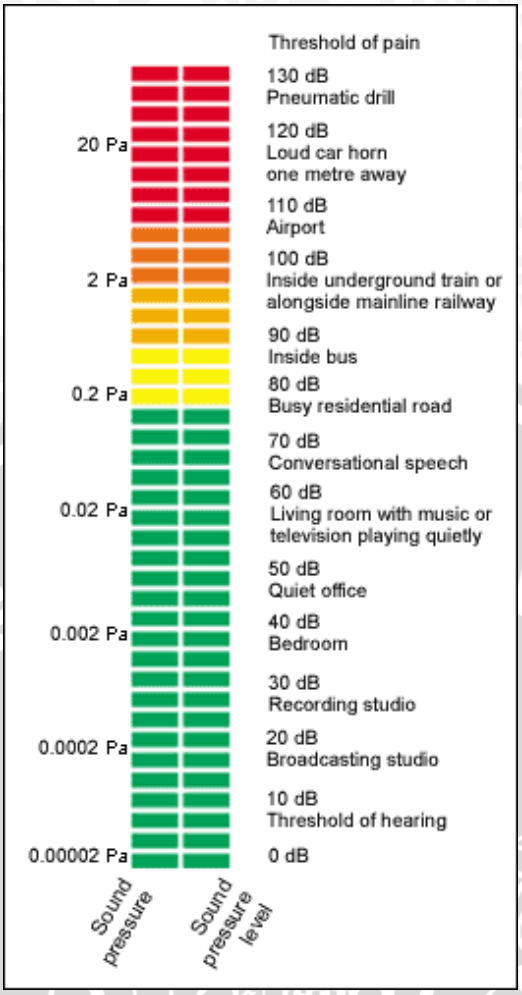
2.2.3 *Sound Pressure Level* (SPL)

Tingkat tekanan suara (SPL) menunjukkan seberapa besar perubahan tekanan (*pascall*/Pa) yang dialami oleh medium, umumnya udara dari kondisi setimbangnya. Tingkat tekanan suara dihitung dari nilai tekanan suara efektif (P_{rms}) relatif terhadap nilai referensinya (P_{ref}), yang dinyatakan dalam skala logaritmik dalam satuan desibel (dB).

$$SPL = 10log_{10}\left(\frac{{P_{rms}}^2}{{P_{ref}}^2}\right) = 20log_{10}\left(\frac{P_{rms}}{P_{ref}}\right) dB \quad (2\text{-}6)$$

Nilai referensi SPL adalah 0 dB atau setara dengan 20 µPa. Nilai referensi ini didapatkan dari hubungan kasar terhadap tekanan suara minimum yang dapat didengar oleh manusia dengan pendengaran normal. Diketahui bahwa nilai tekanan efektif 1 Pa dapat dinyatakan dengan tingkat tekanan suara 94 dB.

Tingkat tekanan suara ditunjukkan pada Gambar 2.7. Gambar 2.7 menunjukkan bahwa tingkat tekanan suara yang dapat didengar manusia berada pada level 0 dB s/d 130 dB. Semakin besar nilai desibel (dB), maka suara yang didengar semakin nyaring. Level di atas 120 dB umumnya dapat membahayakan pendengaran dan terasa menyakitkan di telinga.



Gambar 2.7 Tingkat tekanan suara

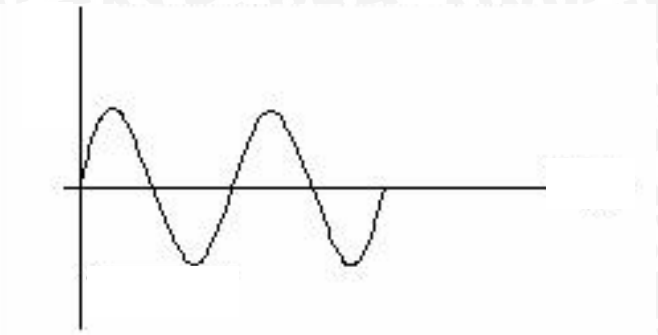
(**Sumber :** http://www.osha.gov)

2.3 Sinyal dan Pemrosesan Sinyal Digital

2.3.1 Pengertian Dasar Sinyal

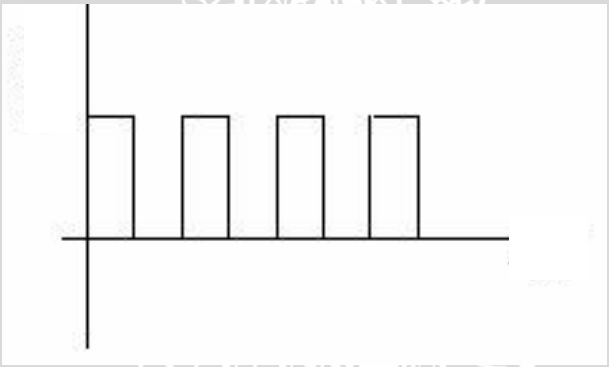
Sinyal secara umum merupakan suatu besaran fisis yang merupakan fungsi waktu, ruangan atau beberapa variabel. Sinyal dikelompokkan menjadi dua jenis, yaitu sinyal analog (kontinyu) dan sinyal digital (diskrit).

Sinyal analog adalah sinyal dalam bentuk gelombang kontinyu yang memiliki nilai pada setiap saat. Sinyal analog umumnya direpresentasikan dalam bentuk gelombang sinusoida. Gambar sinyal analog ditunjukkan pada Gambar 2.8.



Gambar 2.8 Sinyal analog
(**Sumber :** http://www.ustudy.in)

Sinyal digital atau sinyal diskrit merupakan sinyal data dalam bentuk pulsa yang dapat mengalami perubahan tiba-tiba. Sinyal digital hanya memiliki dua keadaan yaitu nol (0) dan satu (1), yang dikenal dengan istilah bit. Gambar sinyal digital ditunjukkan pada Gambar 2.9.



Gambar 2.9 Sinyal digital
(**Sumber :** http://www.ustudy.in)

Pengolahan data pada komputer dilakukan secara digital, melalui sinyal atau tegangan listrik yang diterima dan dikirimkan. Komputer hanya mengenal dua keadaan tegangan, yaitu *on* atau 1 (satu) jika ada tegangan dan *off* atau 0 (nol) jika tidak ada tegangan. Kombinasi dari keadaan *on* atau *off* ini mampu membuat komputer melakukan banyak hal, baik dalam pemrosesan huruf, gambar, suara, bahkan video dalam format digital.

Sinyal digital adalah bentuk *sampling* dari sinyal analog, yang merupakan hasil dari pemrosesan sinyal digital pada komputer. Sinyal digital akan disimpan dalam bentuk data biner atau heksadesimal. Besarnya nilai level sinyal digital dibatasi oleh lebar/jumlah bit (*bandwidth*). Jumlah kemungkinan nilai level yang terbentuk oleh kombinasi n bit adalah sebanyak 2^n.

Sinyal digital memiliki berbagai kelebihan dibandingkan dengan sinyal analog, antara lain yaitu:

- Mampu mengirimkan informasi dengan kecepatan tinggi.
- Penggunaan yang berulang-ulang terhadap informasi tidak mempengaruhi kualitas dan kuantitas informasi itu sendiri,
- Informasi dapat dengan mudah disimpan, diproses dan dimodifikasi ke dalam berbagai bentuk,
- Dapat memproses informasi dalam jumlah besar.
- Lebih kebal terhadap *noise* karena bekerja pada level 0 dan 1.
- Lebih kebal terhadap perubahan temperatur.

2.3.2 Pemrosesan Sinyal Digital

Komputer hanya mampu mengenal sinyal dalam bentuk digital, yaitu tegangan dalam bentuk angka nol (0) dan satu (1). Dengan kecepatan perhitungan yang dimiliki, komputer mampu melihat angka-angka 0 dan 1 ini menjadi kumpulan bit-bit, dan menerjemahkannya menjadi informasi yang bernilai.

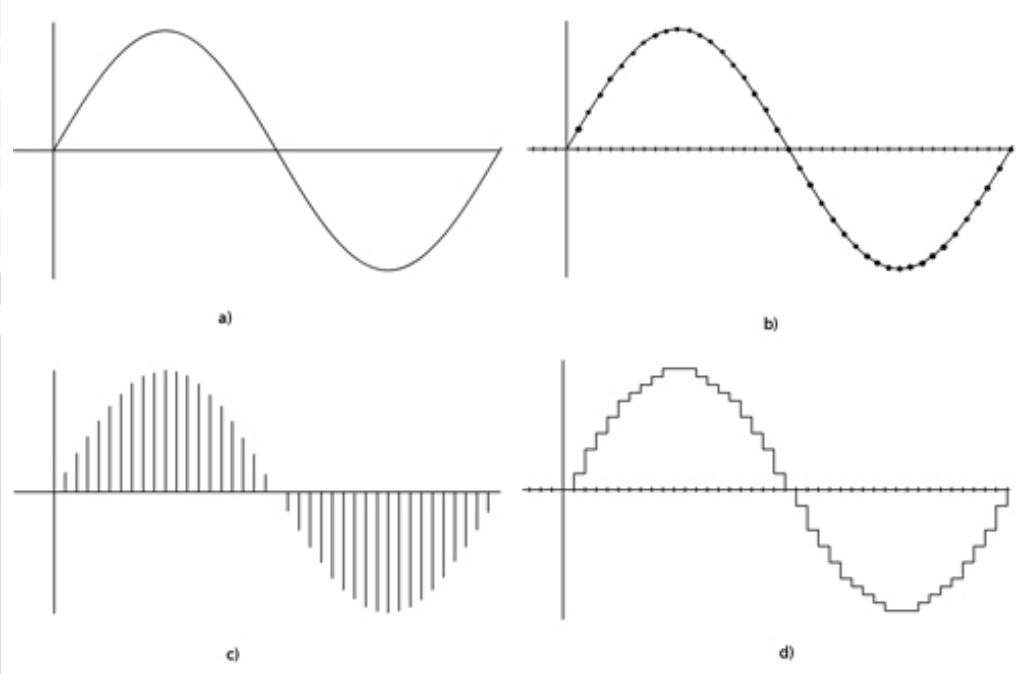
Agar sinyal analog dapat diolah oleh komputer, maka terlebih dahulu harus dilakukan perubahan menjadi sinyal digital. Pemrosesan sinyal analog menjadi bit-bit digital terdiri atas 3 tahapan yang berurutan yaitu *sampling, quantizing dan encoding*.

Pada proses *sampling* terjadi pencuplikan sinyal analog terhadap interval waktu tertentu, sehingga dihasilkan bentuk sinyal waktu diskrit dengan sejumlah nilai yang disebut *sample*. Proses ini harus memenuhi kriteria Teorema *Nyquist*, yang menyatakan bahwa untuk mendapatkan hasil *sampling* yang sesuai dengan sinyal asal (sinyal analog), dibutuhkan frekuensi *sampling* paling sedikit dua kali frekuensi sinyal asal tersebut.

Quantizing adalah proses pembandingan nilai atau level dari tiap-tiap diskrit sinyal hasil *sampling* dengan tetapan level tertentu. Besar tetapan level tergantung pada resolusi kuantisasi dalam bentuk biner (bit). Nilai-nilai diskrit sinyal akan dinaikkan atau diturunkan mengikuti level kuantisasi yang memungkinkan, sehingga terdapat perbedaan nilai dengan sinyal asalnya. Perbedaan nilai yang terjadi akibat kuantisasi disebut dengan *quantization error* (*quantization noise*).

Setelah memiliki nilai tetapan tertentu, tiap-tiap diskrit sinyal kemudian diubah menjadi kombinasi bilangan biner (*encoding*). Kumpulan dari kombinasi bilangan biner ini merupakan data digital yang berisi informasi dari sinyal asal, yang selanjutnya akan diolah oleh komputer.

Langkah pemrosesan sinyal analog menjadi sinyal digital ditunjukkan pada Gambar 2.10.



Gambar 2.10 Langkah pemrosesan sinyal analog menjadi sinyal digital
(**Sumber :** http://www.jiscdigitalmedia.ac.uk)

Gambar 2.10 (a) adalah sinyal analog yang akan diproses. Titik-titik *sample* dengan interval waktu tertentu ditunjukkan pada Gambar 2.10 (b). Nilai amplitudo hasil *sampling* (Gambar 2.10 (c)) kemudian dibandingkan sesuai nilai level yang memungkinkan melalui proses *quantizing* (Gambar 2.10 (d)).

Beberapa parameter yang perlu diperhatikan pada pemrosesan sinyal digital antara lain:

1. Laju pencuplikan (*sampling rate*)

   Laju pencuplikan adalah banyaknya jumlah pencuplikan dalam satu satuan waktu. Makin besar laju pencuplikan, maka hasil *sampling* makin mendekati bentuk sinyal aslinya. Untuk *sampling rate* sebesar 44.1 kHz, dalam satu detik dilakukan pencuplikan sebanyak 44100 kali.

2. Frekuensi *Nyquist*

   Besar frekuensi *Nyquist* adalah setengah dari *sampling rate*. Frekuensi *Nyquist* adalah frekuensi maksimum yang dapat dideteksi sesuai dengan besar *sampling rate* yang digunakan. Hal ini dikarenakan untuk mengidentifikasi satu periode gelombang sinusoida dibutuhkan paling sedikit dua titik *sample* (puncak dan lembah).

3. Banyak bit (*bit depth*)

   Banyaknya bit yang digunakan untuk merepresentasikan level kuantisasi atau nilai amplitudo dinamakan *bit depth*. Semakin besar jumlah bit yang digunakan, maka jangkauan level amplitudo akan semakin halus dengan nilai interval yang semakin kecil. Untuk *bit depth* sebesar 16 bit, jumlah level yang memungkinkan adalah 65536 (2^{16}) level, dengan jangkauan nilai level 0 s/d 65535.

Semakin besar nilai *sampling rate* dan *bit depth*, maka tingkat akurasi sinyal digital terhadap sinyal asal (sinyal analog) menjadi semakin besar dan kualitas sinyal digital semakin baik. Akan tetapi pemrosesannya membutuhkan penyimpanan data dalam jumlah yang besar. Jika pemrosesan sinyal dilakukan pada perangkat yang canggih, penyimpanan data dalam jumlah besar tidak akan menjadi masalah. Sedangkan untuk pemrosesan sinyal pada frekuensi tingkat rendah, nilai *sampling rate* yang besar tidak diperlukan.

2.4 Pemrosesan Audio Digital

Sinyal audio hanya dapat diolah oleh komputer dalam bentuk digital (audio digital). Pemrosesan sinyal audio dari analog menjadi digital dilakukan melalui proses perekaman suara digital (*digital recording*). Beberapa hal yang perlu diketahui sebelum melakukan *digital recording* antara lain yaitu format PCM, *audio quality* dan transduser.

2.4.1 *Pulse Code Modulation* (PCM)

Untuk mendapatkan informasi sinyal dalam bentuk data digital secara utuh, perekaman suara dilakukan dalam format PCM (*Pulse Code Modulation*). PCM adalah sebuah metode yang digunakan untuk mengubah sinyal analog ke dalam bentuk digital. Metode PCM membentuk representasi digital dari sebuah sinyal analog, dimana konversi sinyal dilakukan melalui proses *sampling*, *quantizing* dan *encoding*. Hasil

konversi ini disebut dengan format PCM. Format PCM bersifat tidak terkompresi, dan merupakan bentuk standar dari audio digital pada komputer, CD, DVD maupun penggunaan lain seperti sistem telepon digital.

2.4.2 *Audio Quality*

Kualitas suara audio dari hasil perekaman dipengaruhi oleh *channel*, *sampling rate* dan *bit depth*, yang terangkum dalam kualitas audio (*audio quality*). Daftar kualitas audio ditunjukkan pada Tabel 2.2.

Tabel 2.2 Audio quality yang digunakan pada digital recording

No	*Audio Quality*	*Channel*	*Sampling Rate*	*Bit Depth*
1	*Telephone Quality*	*Mono*	8 kHz	8 bit
2	*CD Quality*	*Stereo*	44.1 kHz	16 bit
3	*DVD Quality*	*Stereo*	48 kHz	16 bit
4	*Studio Quality*	*Stereo*	44.1 kHz / 48 kHz	24 bit

(**Sumber :** http://www.dspguide.com)

Channel merupakan jumlah kanal/saluran masukan pada media *input* sinyal audio, mulai dari satu kanal (*mono*), dua kanal (*stereo*) bahkan lebih. Media *input* yang digunakan umumnya adalah *microphone*.

Sampling rate dan *bit depth* merupakan spesifikasi pemrosesan sinyal audio analog menjadi digital. Media untuk pemrosesan sinyal adalah *Analog to Digital Converter* atau ADC.

Semakin tinggi kualitas audio yang digunakan, maka kualitas suara hasil perekaman semakin jernih dan semakin mendekati suara aslinya. Kualitas audio digunakan sesuai fungsinya masing-masing tergantung dari besar frekuensi sinyal yang akan diproses dan kemampuan perangkat *recording*.

2.4.3 Transduser

Transduser didefinisikan sebagai suatu piranti yang dapat mengubah suatu energi ke bentuk energi yang lain. Tranduser dibagi menjadi dua, yaitu transduser *input* dan transduser *output*. Transduser *input* dapat mengubah energi non-elektrik menjadi energi elektrik. Sebaliknya transduser *output* dapat mengubah energi elektrik menjadi energi non-elektrik. Transduser yang digunakan dalam pemrosesan audio digital antara lain yaitu *microphone* (transduser *input*) dan *loudspeaker* (transduser *output*).

2.4.3.1 *Microphone*

2.4.3.1.1 Prinsip Kerja *Microphone*

Microphone berfungsi untuk mengubah energi akustik (gelombang suara) menjadi energi elektrik (sinyal audio). *Microphone* memiliki diafragma yang akan bergetar ketika ditumbuk gelombang suara. Getaran ini akan diubah ke dalam arus listrik yang kemudian menjadi sinyal audio.

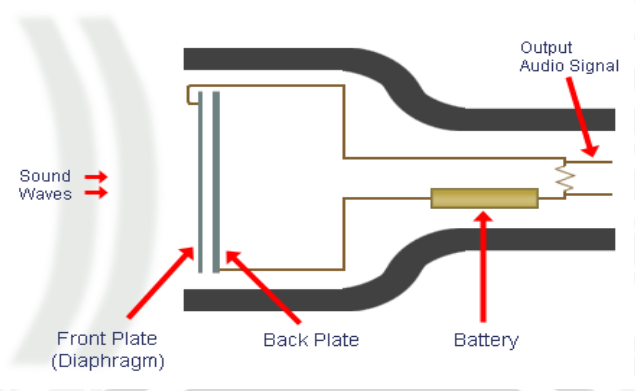
Arus listrik yang dibangkitkan oleh *microphone* sangat rendah. Sinyal audio pada umumnya diukur dalam millivolt (*level mic*). Sebelum dapat digunakan untuk kebutuhan, sinyal dikuatkan hingga pada *line level* (0,5 volt s/d 2 volt). *Line level* merupakan kuat sinyal standar yang digunakan oleh peralatan pemroses audio seperti CD *player* atau *voice recorder*. Penguatan dilakukan dengan menggunakan *amplifier* untuk menaikkan tegangan sinyal hingga pada *level line*.

2.4.3.1.2 *Microphone* Kondenser

Kondenser atau kapasitor adalah suatu komponen elektronik yang menyimpan energi dalam bentuk medan elektrostatik. *Mic* kondenser menggunakan elemen dasar kapasitor untuk mengubah energi akustik menjadi energi elektrik. Agar *mic* dapat berfungsi dibutuhkan tegangan pada ujung-ujung kapasitor yang disediakan oleh baterai dalam *mic* atau sumber tegangan dari luar.

Pada *mic* kondenser terdapat kapasitor yang terdiri atas dua keping plat atau piringan yang keduanya mempunyai tegangan. Salah satu plat ini terbuat dari materi yang sangat ringan yang bertindak sebagai diafragma dan sensitif terhadap gelombang suara. Diafragma tersebut akan bergetar jika ada gelombang suara yang datang sehingga mengubah jarak antara dua plat tersebut serta mengubah nilai kapasitansi. Jika kedua plat berdekatan satu sama lain, maka nilai kapasitansi bertambah dan terjadi muatan arus. Jika kedua plat berjauhan maka nilai kapasitansi berkurang dan terjadi pengosongan muatan arus.

Mic kondenser umumnya memiliki sensitivitas yang lebih tinggi dari jenis *mic* lainya dan dapat menghasilkan suara yang lebih *smooth/natural*, khususnya pada frekuensi tinggi. *Mic* kondenser mampu mencapai respon frekuensi yang *flat* dan memiliki *range* frekuensi yang lebih luas. Karena hanya menggunakan dua plat yang dapat disesuaikan ukurannya, *mic* kondenser dapat berukuran sangat kecil dan ringan.



Gambar 2.11 Potongan melintang *microphone* kondenser

(**Sumber :** Sri Waluyanti dkk, 2008)

2.4.3.1.3 *Microphone* Kondenser Elektret

Microphone kondenser elektret merupakan jenis *mic* kondenser dengan kapasitor sebagai elemen dasarnya. Perbedaannya adalah *mic* kondenser elektret menggunakan jenis kapasitor khusus yang memiliki tegangan permanen. Bahan dasar kapasitor yang digunakan yaitu semacam magnet permanen yang tidak membutuhkan sumber tegangan dari luar. Oleh karena itu *mic* kondenser elektret biasanya dilengkapi dengan *pre-amplifier* yang masih membutuhkan daya.

Mic ini bersifat *omni-directional* atau mengambil suara dari segala arah. *Mic* kondenser elektret banyak digunakan dalam berbagai perangkat elektronik sepeti ponsel, *laptop*, *headphone*, dan lain-lain.

Gambar 2.12 *Microphone* kondenser elektret

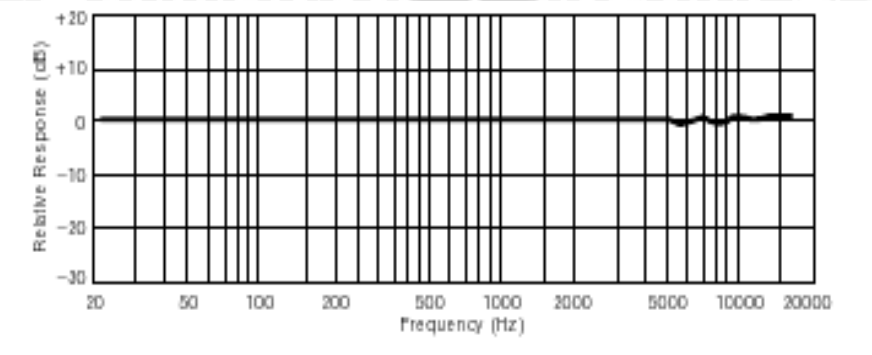
(**Sumber :** http:// www.icmercato.it)

2.4.3.1.4 Respon Frekuensi *Microphone*

Respon frekuensi *microphone* didefinisikan sebagai rentang suara (dari frekuensi terendah hingga tertinggi) yang dapat dihasilkan dan variasinya di antara rentang tersebut. Kurva respon frekuensi menunjukkan keberhasilan mereproduksi frekuensi yang bervariasi. Respon frekuensi yang ideal memiliki bentuk datar (*flat*)

yang berarti bahwa *microphone* mempunyai kepekaan yang sama untuk semua frekuensi, tidak ada frekuensi di kuatkan atau dikurangi.

Respon frekuensi *microphone* bervariasi, tergantung dari jenis dan kualitas bahan yang digunakan. Berikut ini adalah respon frekuensi dari *microphone* kondenser elektret secara umum.



Gambar 2.13 Respon frekuensi *microphone* kondenser elektret secara umum
(**Sumber :** http:// www.panasonic.com)

2.4.3.2 *Loudspeaker*

Fungsi *loudspeaker* adalah mengubah energi elektrik menjadi energi akustik berupa gelombang suara. Komponen utama yang terpasang di dalam *loudspeaker* antara lain yaitu kumparan (lilitan), magnet, dan membran. Getaran membran inilah yang akan menghasilkan suara.

Gambar 2.14 *Loudspeaker*
(**Sumber :** http:// greatplainsaudio.com)

Prinsip kerja *loudspeaker* adalah sebagai berikut.

1. Prinsip dasar

   Pada dasarnya prinsip kerja *loudspeaker* adalah kebalikan dari *microphone*. *Loudspeaker* merupakan perangkat *output* yang mengubah sinyal elektrik

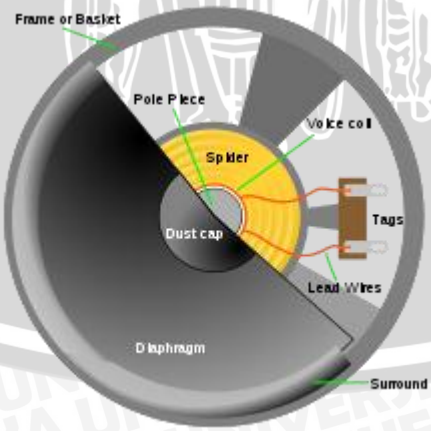
menjadi suara melalui getaran membran sedangkan *microphone* merupakan perangkat *input* yang mengubah suara menjadi sinyal elektrik.

2. Diafragma (membran)

   Gelombang suara diproduksi dengan menggetarkan *cone fleksibel* (diafragma) secara cepat. Getaran tersebut adalah hasil induksi magnet yang mengalir melalui lilitan. *Cone* biasanya terbuat dari kertas yang terhubung pada ujung *suspension* (*surround*). *Surround* sendiri merupakan material fleksibel yang menggerakkan *cone* yang terletak pada bingkai logam (*basket*).

3. Magnet

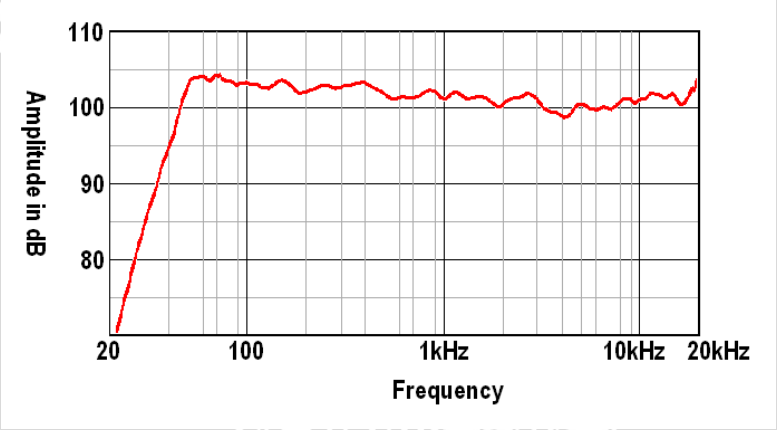
   Elektromagnet diposisikan pada bidang magnet konstan yang terbuat dari magnet permanen. Kedua magnet tersebut, yaitu elektromagnet dan magnet permanen berinteraksi satu sama lain seperti dua magnet yang berhubungan pada umumnya. Kutub positif pada elektromagnet tertarik oleh kutub negatif pada magnet permanen dan kutub negatif pada elektromagnet ditolak oleh kutub negatif magnet permanen. Ketika orientasi kutub elektromagnet bertukar, bertukar pula arah dan gaya tarik-menariknya. Dengan cara seperti ini, arus bolak-balik secara konstan membalikkan dorongan magnet antara *voice coil* (lilitan) dan magnet permanen. Proses inilah yang mendorong *coil* kembali dan begitu seterusnya dengan cepat. Sewaktu *coil* bergerak, coil *tersebut* mendorong dan menarik *cone*, sehingga menggetarkan udara di depan *loudspeaker* dan membentuk gelombang suara.



Gambar 2.15 Penampang *loudspeaker*
(**Sumber :** http:// en.wikipedia.org)

Berdasarkan tanggapan frekuensinya, *loudspeker* digolongkan dalam beberapa jenis, di antaranya:

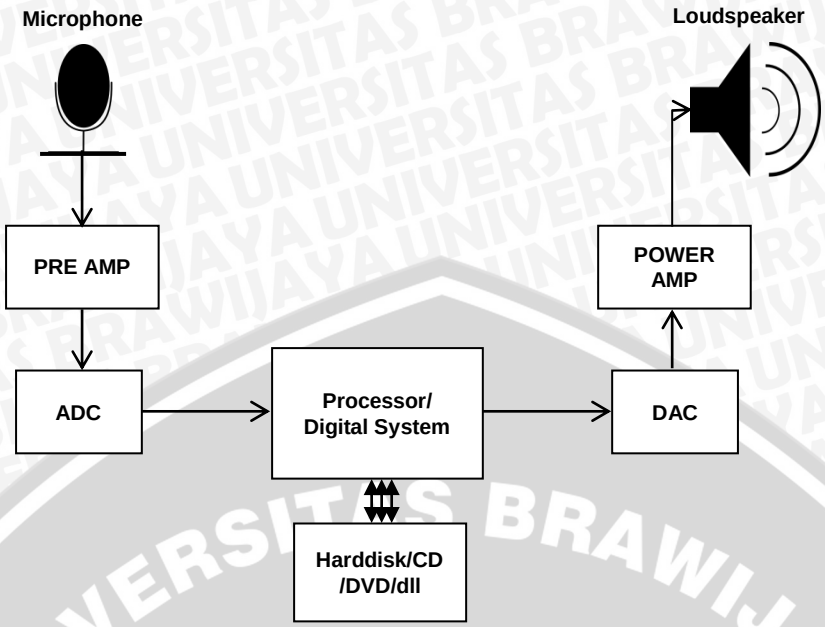
- *Woofer*, *loudspeaker* jenis ini memberikan tanggapan yang sangat baik pada frekuensi rendah (20 Hz s/d 1 Khz). *Woofer* mempunyai ciri berdiameter besar, tekstur tepi *cone* lentur.
- *Midle*, bekerja baik pada frekuensi menengah (1 Khz s/d 10 Khz). Cirinya berdiameter sedang, bagian belakang rangka tertutup rapat.
- *Twitter*, dirancang untuk memberikan tanggapan yang baik pada nada-nada tinggi antara 10 Khz s/d 20 Khz. Cirinya berdiameter kecil, bagian belakang tertutup rapat.
- *Fullrange*, *loudspeaker* jenis ini di rancang untuk dapat memberi tanggapan yang baik pada semua frekuensi suara. Berbentuk sama dengan *woofer*, tetapi tekstur *cone* tidak lentur.



Gambar 2.16 Respon frekuensi *loudspeaker* secara umum (*high-quality*)
(**Sumber :** http:// www.harmonycentral.com)

2.4.4 *Recording* dan *Playback*

Sinyal audio dapat direkam (*record*) melalui media *input* berupa *microphone*, serta dapat diputar kembali (*playback*) melalui media *output* berupa *loudspeaker*. Diagram pemrosesan audio digital secara umum ditunjukkan pada Gambar 2.17.



Gambar 2.17 Diagram pemrosesan audio digital

(**Sumber :** http://www.eeherald.com)

Langkah-langkah perekaman hingga pemutaran suara dijelaskan sebagai berikut.

1. Suara masuk melalui media *input microphone*, dimana *microphone* mengubah variasi-variasi tekanan udara ke dalam nilai-nilai elektrik berupa tegangan.
2. *Pre-amplifier* digunakan untuk menguatkan nilai-nilai tegangan yang didapatkan dari *microphone*.
3. *Analog to Digital Converter* (ADC) mengkonversi tegangan elektrik menjadi data digital dengan berulang kali mengukur perubahan nilai tegangan elektrik.
4. Data digital melewati sistem digital untuk dianalisa, disimpan atau dimanipulasi.
5. Sistem digital membentuk aliran data digital yang dapat diambil dari lokasi penyimpanan data, data hasil manipulasi atau aliran data *real-time* dari perangkat *input*.
6. *Digital to Analog Converter* (DAC) mengkonversi data digital menjadi nilai-nilai tegangan elektrik.
7. *Power Amplifier* digunakan untuk meningkatkan daya (*power*) dari sinyal audio.

8. Media *output* berupa *loudspeaker* mengubah nilai-nilai tegangan elektrik ke dalam variasi-variasi tekanan udara, dan menghasilkan suara yang dapat didengar.

2.4.5 Pembentukan Data Digital

ADC atau *Analog to Digital Converter* menghasilkan data digital berbentuk biner dari nilai tegangan yang terukur. Nilai tegangan diukur setiap interval waktu tertentu atau setiap *sample* yang tercuplik. Pengukuran nlai tegangan dilakukan dengan cara dibandingkan dengan level kuantisasi terdekat. Setiap level kuantisasi memiliki batas atas (HI) dan batas bawah (LO) sesuai resolusi kuantisasi. Untuk *bit depth* sebesar 3 bit, maka jumlah level yang mungkin adalah $2^3 = 8$. Jika tegangan input $V_{min} = 0$ V dan $V_{max} = 4$ V, maka resolusi kuantisasinya adalah 8/(4-0) = 0.5 V. Level kuantisasi untuk 3 bit *bit depth* ditunjukkan pada Tabel 2.3. Jika V *input* < 0V, maka level akan bernilai 0. Sedangkan jika V *input* > 3.75 V, maka level akan bernilai 7.

Tabel 2.3 Level kuantisasi untuk *bit depth* = 3 bit

Count	Vlevel (V)	VthreshHI (V)	VthreshLO (V)
0	0.0	0.25	-0.25
1	0.5	0.75	0.25
2	1.0	1.25	0.75
3	1.5	1.75	1.25
4	2.0	2.25	1.75
5	2.5	2.75	2.25
6	3.0	3.25	2.75
7	3.5	3.75	3.25

(**Sumber :** http://www.ecircuitcenter.com)

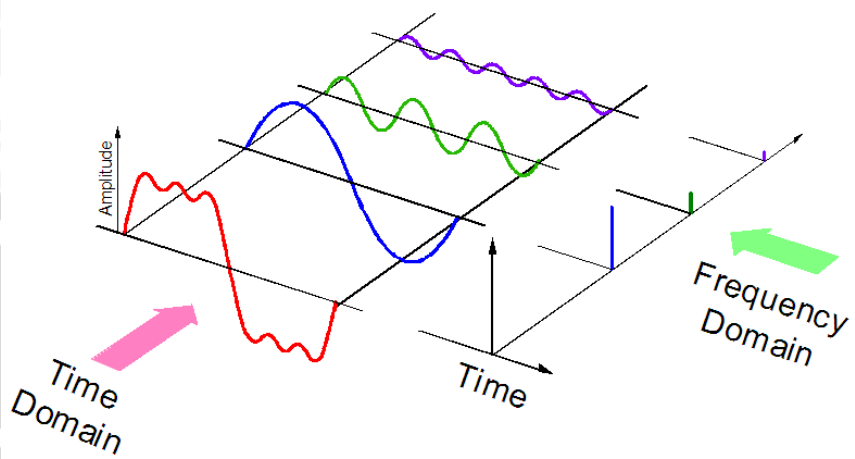
2.5 *Spectrum Analyzer*

Bentuk gelombang dari sebuah sinyal biasanya ditampilkan dalam fungsi waktu. Akan tetapi, sinyal dapat direpresentasikan dalam format lain, yaitu dalam *domain* frekuensi. Setiap bentuk gelombang yang tampil dalam *domain* waktu memiliki representasi yang setara dalam *domain* frekuensi, yang disebut spektrum frekuensi.

Spectrum Analyzer merupakan perangkat yang digunakan untuk mengamati komposisi sebaran spektrum frekuensi dari sebuah sinyal. Dengan melihat sebaran spektrum sinyal, akan didapatkan penjelasan mengenai komponen-komponen dari

sinyal serta performa dari suatu rangkaian atau perangkat yang tidak dapat diukur menggunakan alat lain.

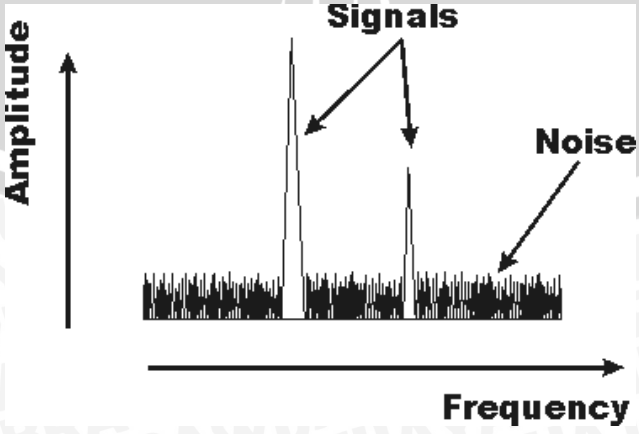
Pada beberapa pengukuran, representasi sinyal menggunakan *domain* frekuensi lebih mudah untuk dianalisis daripada domain waktu. Dengan menggunakan *spectrum analyzer*, akan terlihat komponen frekuensi, amplitudo bahkan *noise* dari sinyal yang dianalisis. *Noise* merupakan suatu sinyal gangguan yang tidak diinginkan.



Gambar 2.18 Diagram hubungan domain waktu dengan domain frekuensi
(**Sumber :** *http://testrf.com)*

Tampilan *domain* frekuensi pada *spectrum analyzer* memiliki dua sumbu. Sumbu vertikal untuk *level* atau amplitudo, sedangkan sumbu horisontal untuk frekuensi. Amplitudo biasanya dinyatakan dalam skala logaritmik dengan satuan desibel (dB), sedangkan frekuensi dinyatakan dalam Hertz (Hz) dengan skala linear.

Format dasar tampilan *spectrum analyzer* dan komponen-komponen sinyal yang ditampilkan ditunjukkan pada Gambar 2.19.



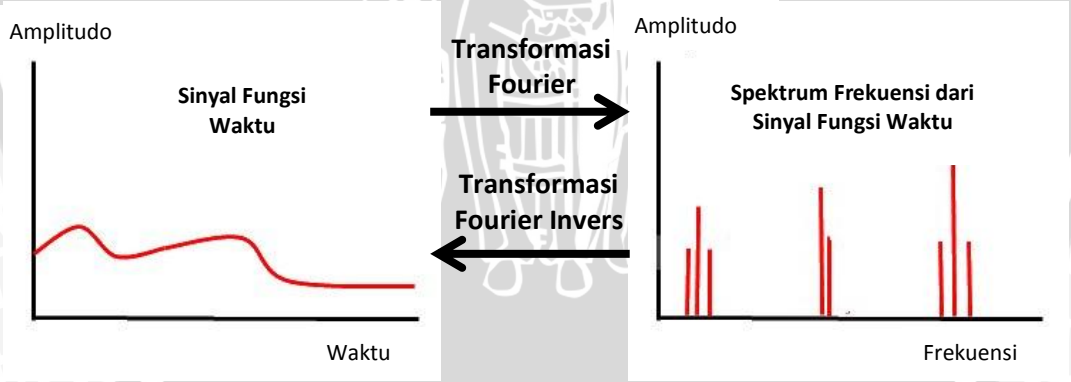
Gambar 2.19 Format dasar tampilan *spectrum analyzer*
(**Sumber :** *http://www.radio-electronics.com*)

Spectrum analyzer secara umum dapat dibagi menjadi dua tipe, yaitu analog dan digital. Tipe digital atau disebut *Fast Fourier Transform (FFT) Spectrum Analyzer* bekerja dengan mencuplik sinyal informasi (*sampling*) yang diterima terhadap suatu interval waktu tertentu dan hasilnya akan diolah secara digital menggunakan metode FFT. Proses digitalisasi ini mempercepat waktu tangkap dan analisis spekrum sinyal.

Audio Spectrum Analyzer termasuk *spectrum analyzer* tipe digital dan menggunakan dasar teknologi FFT. *Audio Spectrum Analyzer* menggunakan teknik frekuensi rendah dan terfokus pada rentang frekuensi audio. Hal ini memungkinkan untuk diterapkan pada komputer yang berdaya rendah, serta dapat juga diimplementasikan pada perangkat *mobile phone*.

2.6 *Fast Fourier Transform* (FFT)

Sinyal dapat digambarkan sebagai gabungan gelombang sinusoida dengan frekuensi dan amplitudo tertentu yang dijumlahkan bersama-sama. Komponen-komponen sinyal berupa frekuensi dan amplitudo dapat ditampilkan dalam *domain* frekuensi, yang disebut dengan spektrum frekuensi. Konversi sinyal dalam *domain* waktu menjadi *domain* frekuensi atau sebaliknya secara matematis dilakukan dengan menggunakan Transformasi *Fourier*.



Gambar 2.20 Ilustrasi Transformasi *Fourier* untuk konversi sinyal (**Sumber :** http://www.alwayslearn.com)

Discrete Fourier Transform (DFT) adalah Transformasi Fourier untuk mengkonversi sinyal diskrit *domain* waktu menjadi *domain* frekuensi. Transformasi sinyal diskrit menggunakan DFT memiliki laju komputasi yang relatif lambat karena membutuhkan perkalian yang kompleks. *Fast Fourier Transform* (FFT) adalah sebuah algoritma yang dibuat untuk mempercepat perhitungan DFT.

2.6.1 *Discrete Fourier Transform* (DFT)

Discrete Fourier Transform atau DFT melakukan konversi pada data-data diskrit dari sinyal dalam *domain* waktu menjadi bentuk spektrum frekuensinya dalam *domain* frekuensi. Dalam penggunaan DFT ditunjukkan bahwa blok data yang dianalisis adalah satu periode *sampling* dari sebuah sinyal periodik yang tidak terbatas.

Persamaan DFT dapat dituliskan sebagai berikut.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn} \quad ; k = 0, 1, 2, \ldots, N-1 \qquad (2\text{-}2)$$

dengan: N = jumlah data *sample*

k = indeks frekuensi

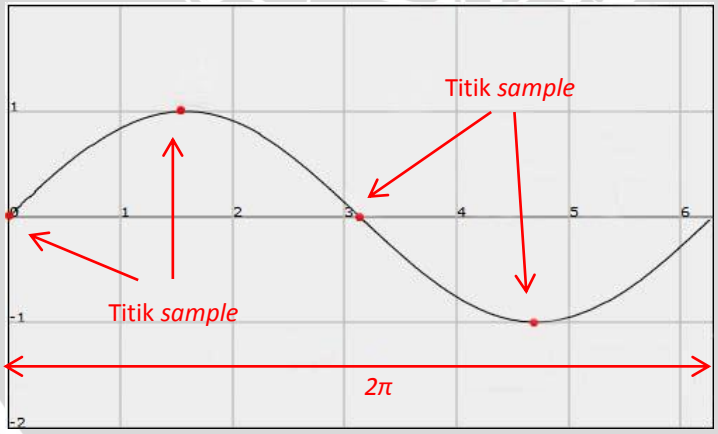
n = indeks waktu

$X(k)$ = nilai pada frekuensi k

$x(n)$ = nilai pada waktu n

$W_N = e^{\frac{-j2\pi}{N}}$ (*twiddle factor*)

Penerapan dari persamaan DFT adalah sebagai berikut. Sebuah gelombang sinusoida sederhana dari sinyal fungsi waktu akan dicuplik sejumlah *N* titik, kemudian dikonversi ke dalam *domain* frekuensi menggunakan DFT. Gelombang sinusoida tersebut ditunjukkan pada Gambar 2.21.



Gambar 2.21 Sinyal fungsi waktu (sebelum perhitungan DFT)
(**Sumber :** http://www.alwayslearn.com)

Pencuplikan dilakukan sebanyak 4 titik ($N = 4$), dengan periode dari gelombang yang dicuplik adalah 2π. Sehingga jarak antar titik *sample* adalah $T/N = 2\pi/4 = \pi/2$. Fungsi *sample* gelombang dapat dinyatakan dengan persamaan berikut.

$$x(n) = \sin(\frac{\pi}{2}n) \quad ; n = 0, 1, 2, \ldots, N-1 \qquad (2\text{-}3)$$

Titik *sample* terakhir adalah titik *sample* sebelum *2π*. Hal ini dikarenakan gelombang dianggap sebagai pola yang berulang sehingga titik *sample* berikutnya akan kembali pada posisi awal.

Jika diketahui waktu yang dibutuhkan untuk mencuplik satu periode gelombang adalah 0.1 detik, maka dapat diketahui frekuensi dari gelombang adalah 10 Hz. Karena waktu *sampling* setiap titik adalah *T/N = 0.1/4 = 0.025 detik*, maka diperoleh nilai frekuensi sampling (*sampling rate)* yang digunakan yaitu *1/0.025 = 40 Hz*.

Masing-masing nilai titik *sample* adalah sebagai berikut.

$$untuk\ n = 0,\ x(0) = \sin(0) = 0$$
$$untuk\ n = 1,\ x(1) = \sin(\frac{\pi}{2}) = 1$$
$$untuk\ n = 2,\ x(2) = \sin(\pi) = 0$$
$$untuk\ n = 3,\ x(3) = \sin(\frac{3\pi}{2}) = -1 \qquad (2\text{-}4)$$

Perhitungan DFT membutuhkan nilai W_N^{kn}. Nilai W_N^{kn} untuk $N = 4$ dan $kn = 0$ *s/d 4* adalah sebagai berikut.

$$W_4^{kn} = e^{\frac{-j2\pi kn}{4}} = \cos(\frac{-2\pi kn}{4}) + j\sin(\frac{-2\pi kn}{4})$$
$$untuk\ kn = 0,\ W_4^0 = \cos(0) + j\sin(0) = 1$$
$$untuk\ kn = 1,\ W_4^1 = \cos(\frac{\pi}{2}) + j\sin(\frac{-\pi}{2}) = -j$$
$$untuk\ kn = 2,\ W_4^2 = \cos(\pi) + j\sin(-\pi) = -1$$
$$untuk\ kn = 3,\ W_4^3 = \cos(\frac{3\pi}{2}) + j\sin\left(\frac{-3\pi}{2}\right) = j \qquad (2\text{-}5)$$

Sedangkan nilai W_N^{kn} untuk nilai $kn \geq 4$ akan mengalami perputaran siklus ke empat nilai pertama. Jadi nilai $W_N^0 = W_N^4 = W_N^8$, $W_N^1 = W_N^5 = W_N^9$ dan seterusnya. Nilai W_N^{kn} yang digunakan untuk nilai $kn \geq 4$ adalah *modulus 4* dari *kn*.

Penguraian persamaan *4-Point* DFT adalah sebagai berikut.

$$X(k) = x(0)W_4^{(0)(k)} + x(1)W_4^{(1)(k)} + x(2)W_4^{(2)(k)} + x(3)W_4^{(3)(k)} \qquad (2\text{-}6)$$

Hasil perhitungan *4-Point* DFT untuk masing-masing nilai *sample* adalah sebagai berikut.

$$X(0) = x(0)W_4^{(0)(0)} + x(1)W_4^{(1)(0)} + x(2)W_4^{(2)(0)} + x(3)W_4^{(3)(0)}$$
$$X(0) = (0)(1) + (1)(1) + (0)(1) + (-1)(1) = 0$$

$$X(1) = x(0)W_4^{(0)(1)} + x(1)W_4^{(1)(1)} + x(2)W_4^{(2)(1)} + x(3)W_4^{(3)(1)}$$
$$X(1) = (0)(1) + (1)(-j) + (0)(-1) + (-1)(j) = -2j$$

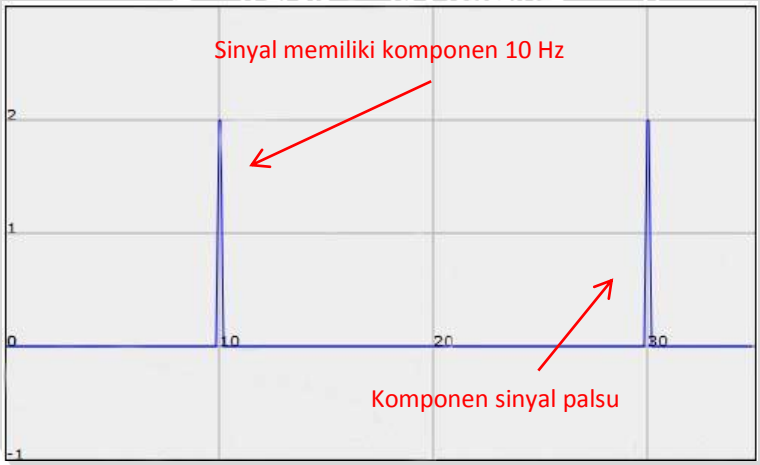
$$X(1) = x(0)W_4^{(0)(2)} + x(1)W_4^{(1)(2)} + x(2)W_4^{(2)(2)} + x(3)W_4^{(3)(2)}$$
$$X(1) = (0)(1) + (1)(-1) + (0)(1) + (-1)(-1) = 0$$

$$X(1) = x(0)W_4^{(0)(3)} + x(1)W_4^{(1)(3)} + x(2)W_4^{(2)(3)} + x(3)W_4^{(3)(3)}$$
$$X(1) = (0)(1) + (1)(j) + (0)(-1) + (-1)(-j) = 2j \qquad (2\text{-}7)$$

Setiap nilai *X(k)* merujuk pada frekuensi tertentu. Dengan *sampling rate* sebesar 40 Hz dan jumlah *sample* sebanyak 4 titik, maka nilai interval frekuensi adalah *40/4 = 10 Hz*.

Besar frekuensi *Nyquist* adalah setengah dari *sampling rate* yaitu *40/2 = 20 Hz*. Frekuensi *Nyquist* merupakan titik potong validasi data dari hasil perhitungan DFT. Jadi semua nilai *X(k)* pada frekuensi lebih dari frekuensi *Nyquist* tidak bernilai valid dan bukan komponen frekuensi sinyal yang sebenarnya.

Hasil spektrum frekuensi dari sinyal fungsi waktu setelah dilakukan perhitungan DFT ditunjukkan pada Gambar 2.22.



Gambar 2.22 Spektrum frekuensi dari sinyal fungsi waktu (setelah perhitungan DFT)
(**Sumber :** http://www.alwayslearn.com)

Dari Gambar 2.22 terlihat bahwa hasil perhitungan DFT memunculkan sebuah komponen frekuensi yang ada pada spektrum frekuensi sinyal yaitu 10 Hz. Frekuensi sinyal telah diketahui sebelumnya yaitu 10 Hz, jadi perhitungan DFT terbukti. Sedangkan frekuensi 30 Hz yang terlihat pada gambar adalah komponen frekuensi sinyal yang tidak valid atau palsu.

Dari hasil perhitungan DFT dapat diperoleh nilai-nilai komponen berupa amplitudo dan fasa pada frekuensi tertentu yang merupakan karakteristik dari suatu sinyal. Nilai amplitudo (*magnitude*) dapat diperoleh menggunakan persamaan sebagai berikut.

$$|X(k)| = \sqrt{X_{real}(k)^2 + X_{imaginer}(k)^2} \quad (2\text{-}8)$$

Sedangkan nilai fasa dapat diperoleh menggunakan persamaan sebagai berikut.

$$\theta(k) = \tan^{-1} \frac{X_{imaginer}(k)}{X_{real}(k)} \quad (2\text{-}9)$$

2.6.2 Pembentukan algoritma FFT *Decimation In Time*

Algoritma FFT dibentuk dengan melakukan penguraian terhadap DFT. Karena data *input* adalah data diskrit dari sinyal fungsi waktu, maka penguraian dilakukan dalam fungsi waktu atau disebut dengan FFT *Decimation In Time*.

DFT diuraikan menjadi dua penjumlahan dengan ukuran yang sama atau setengah dari jumlah data input *N*. Penjumlahan dibagi dua berdasarkan urutan genap dan ganjil. Penguraian terhadap DFT dengan penjumlahan urutan genap (*E*) dan penjumlahan urutna ganjil (*O*) dapat dituliskan sebagai berikut.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, \mathrm{e}^{\frac{-j2\pi kn}{N}} = E + O \quad (2\text{-}10)$$

Untuk penjumlahan urutan genap (*E*) dilakukan subtitusi *2n* ke dalam *n*. Penjumlahan dengan urutan genap sejumlah N/2 dapat dituliskan sebagai berikut.

$$E = \sum_{n=0}^{\frac{N}{2}-1} x(2n)\, \mathrm{e}^{\frac{-j2\pi k(2n)}{N}} = \sum_{n=0}^{\frac{N}{2}-1} x(2n)\, \mathrm{e}^{\frac{-j2\pi kn}{(\frac{N}{2})}} \quad (2\text{-}11)$$

Sedangkan untuk penjumlahan urutan ganjil (*O*) dilakukan substitusi *2n+1* ke dalam *n*. Penjumlahan dengan urutan ganjil sejumlah N/2 dapat dituliskan sebagai berikut.

$$O = \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\, \mathrm{e}^{\frac{-j2\pi k(2n+1)}{N}}$$

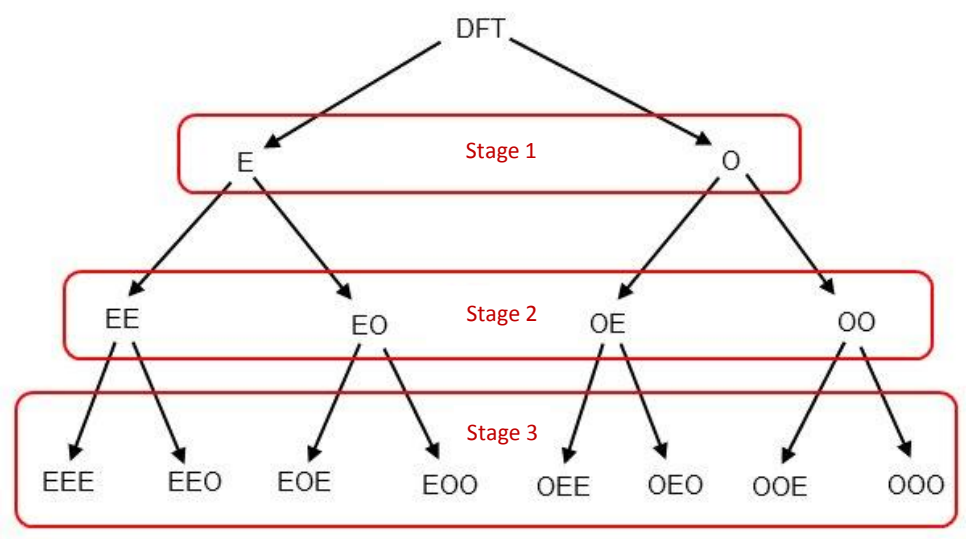
$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\, \mathrm{e}^{\frac{-j2\pi kn}{(\frac{N}{2})}}\, \mathrm{e}^{\frac{-j2\pi k}{N}}$$

$$= e^{\frac{-j2\pi k}{N}} \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\, e^{\frac{-j2\pi kn}{(\frac{N}{2})}} \qquad (2\text{-}12)$$

Dengan mensubstitusi persamaan $W_N = e^{\frac{-j2\pi}{N}}$ ke dalam persamaan (2-10), (2-11) dan (2-12) maka didapatkan persamaan penguraian level pertama DFT sebagai berikut.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n)\, W_{\frac{N}{2}}^{kn} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\, W_{\frac{N}{2}}^{kn} \qquad (2\text{-}13)$$

Penguraian DFT dilanjutkan dengan membagi dua penguraian level pertama DFT menjadi penjumlahan dengan urutan genap dan ganjil (penguraian level kedua), dan seterusnya. Penguraian DFT terus dilakuakn hingga hanya terdapat satu nilai data pada setiap penjumlahan. Ilustrasi penguraian DFT ditunjukkan pada Gambar 2.23.



Gambar 2.23 Ilustrasi penguraian DFT

(**Sumber :** http://www.alwayslearn.com)

Dari Gambar 2.23 dapat disimpulkan bahwa algoritma *N-Point* FFT memerlukan data *input N* sejumlah perpangkatan dua atau $N = 2^v$. Untuk setiap *N* data *input*, penguraian DFT akan dilakukan sebanyak *v* level (*stage*).

2.6.3 *Bit Reversed*

Proses penguraian atau *decimation* yang dilakukan pada pembentukan algoritma FFT mengakibatkan data *input* tersusun secara *bit-reversed*. Susunan *bit-reversed* atau pembalikan bit untuk $N = 8$ ditunjukkan pada Tabel 2.4. Pola *bit-reversed*

data *input* berlaku untuk semua nilai *N* pada *N-Point* FFT.

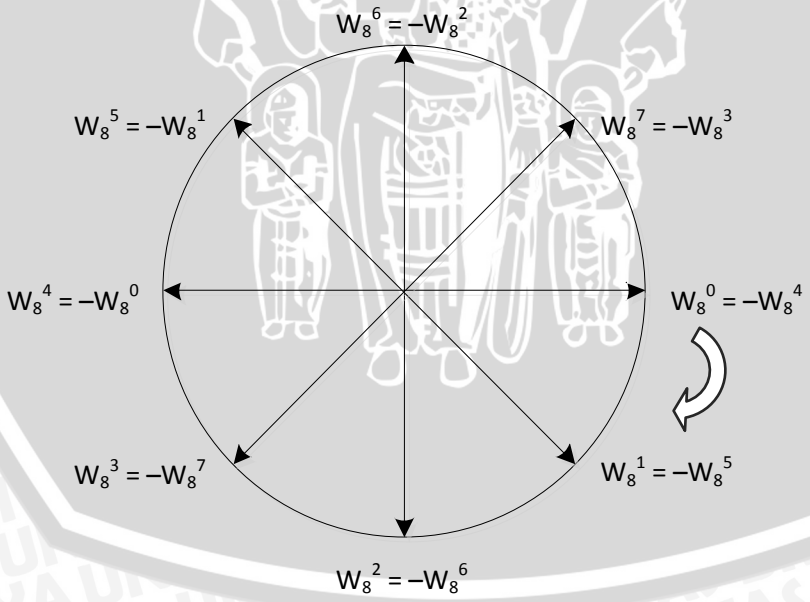
Tabel 2.4 Susunan *bit-reversed* data *input* untuk N = 8

No	Urutan data *input* (normal)	Urutan data *input* (*bit-reserved*)
1	000 (0)	000 (0)
2	001 (1)	100 (4)
3	010 (2)	010 (2)
4	011 (3)	110 (6)
5	100 (4)	001 (1)
6	101 (5)	101 (5)
7	110 (6)	011 (3)
8	111 (7)	111 (7)

(**Sumber :** Harlianto Tanudjaja, 2007)

2.6.4 *Twiddle Factor*

Twiddle factor dapat digambarkan sebagai sebuah rotasi lingkaran, yang berotasi berdasarkan penambahan dari jumlah data *input N*. *Twiddle factor* untuk *N = 8* ditunjukkan pada Gambar 2.24.



Gambar 2.24 Lingkaran *twiddle factor* untuk N = 8

(**Sumber :** Harlianto Tanudjaja, 2007)

Nilai *twiddle factor* adalah sama untuk setiap vektor rotasi yang mengalami perputaran.

$$W_8^0 = W_8^8 = W_8^{16} = W_8^{24} = \ldots$$
$$W_8^1 = W_8^9 = W_8^{17} = W_8^{25} = \ldots \quad (2\text{-}14)$$

Sedangkan nilai *twiddle factor* antara vektor yang berbeda 180 derajat adalah negatif satu sama lain.

$$W_8^0 = -W_8^4 \; ; \; W_8^8 = -W_8^{12} \; ; \; \ldots$$
$$W_8^1 = -W_8^5 \; ; \; W_8^9 = -W_8^{13} \; ; \; \ldots \quad (2\text{-}15)$$

2.6.5 Diagram *Butterfly*

Diagram *butterfly* merupakan representasi diagramatis dari algoritma FFT. Diagram *butterfly* dibuat berdasarkan teorema penguraian DFT, susunan *bit-reversed* data *input* serta sifat istimewa *twiddle factor*.

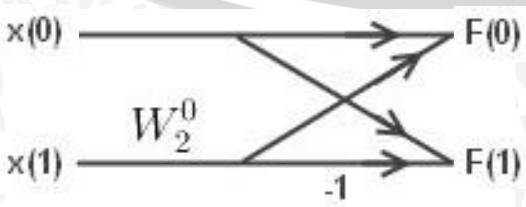
Diagram dasar *butterfly* dibuat dari *2-Point* FFT dengan menggunakan penguraian DFT level pertama. Dengan mensubstitusi nilai *N = 2* pada persamaan (2-12) maka didapatkan persamaan sebagai berikut.

$$X(k) = \sum_{n=0}^{0} x(2n)\, \mathrm{e}^{\frac{-j2\pi kn}{(\frac{2}{2})}} + W_2^k \sum_{n=0}^{0} x(2n+1)\, \mathrm{e}^{\frac{-j2\pi kn}{(\frac{2}{2})}}$$
$$= x(0)\, \mathrm{e}^0 + W_2^k\, x(1)\, \mathrm{e}^0$$
$$= x(0) + W_2^k\, x(1) \quad (2\text{-}16)$$

Dengan mensubstitusi sifat *twiddle factor* $W_2^1 = -W_2^0$, maka didapatkan nilai *X(k)* untuk *k = 0* dan *k = 1* sebagai berikut.

$$X(0) = x(0) + W_2^0\, x(1)$$
$$X(1) = x(0) + W_2^1\, x(1) = x(0) - W_2^0\, x(1) \quad (2\text{-}17)$$

Diagram dasar *butterfly* (*2-point butterfly*) memiliki dua nilai *input* dan menghasilkan dua nilai *output* seperti yang ditunjukkan pada Gambar 2.25.



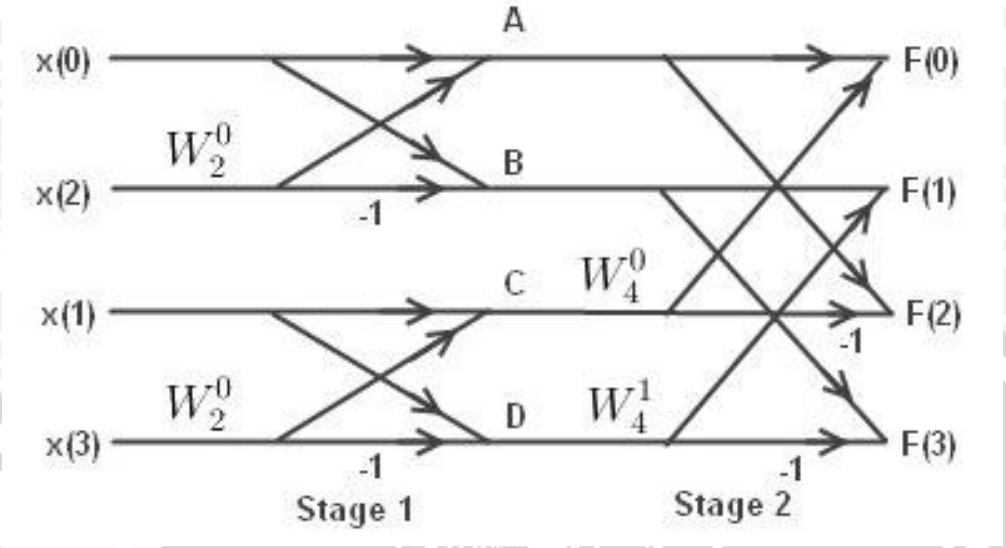
F(0)=X(0) = x(0) + W_2^0 x(1)

F(1)=X(1) = x(0) - W_2^0 x(1)

Gambar 2.25 Diagram dasar *butterfly*

(**Sumber :** Hwei P. Hsu, 1995)

Diagram *butterfly* untuk *4-Point* FFT (*4-point butterfly*) ditunjukkan pada Gambar 2.26. Diagram ini didapatkan dengan cara menggabungkan 2 blok *butterfly* dari *2-Point* FFT menjadi 1 blok *butterfly 4-Point* FFT. Dari Gambar 2.26 dapat dilihat bahwa data *input* tersusun secara *bit-reversed*.



Gambar 2.26 Diagram *butterfly* untuk *4-Point* FFT
(**Sumber :** Hwei P. Hsu, 1995)

Persamaan *4-Point* FFT dari diagram *butterfly* di atas adalah sebagai berikut.

$Stage\ 1$: $A = x(0) + W_2^0\, x(2)$ $\quad C = x(1) + W_2^0\, x(3)$

$B = x(0) - W_2^0 x(2)$ $\quad D = x(1) - W_2^0\, x(3)$

$Stage\ 2$:

$$F(0) = X(0) = A + W_4^0 C = x(0) + W_2^0\, x(2) + W_4^0(x(1) + W_2^0\, x(3))$$
$$F(1) = X(1) = B + W_4^1 D = x(0) - W_2^0\, x(2) + W_4^1(x(1) - W_2^0\, x(3))$$
$$F(2) = X(2) = A - W_4^0 C = x(0) + W_2^0\, x(2) - W_4^0(x(1) + W_2^0\, x(3))$$
$$F(3) = X(3) = B - W_4^1 D = x(0) - W_2^0\, x(2) - W_4^1(x(1) - W_2^0\, x(3)) \quad (2\text{-}18)$$

2.6.6 Perbandingan Komputasi DFT dan FFT

DFT membutuhkan *N* operasi perkalian untuk setiap data sejumlah *N*, sehingga komputasi DFT adalah $O(N^2)$. Sedangkan FFT hanya membutuhkan *N* operasi perkalian (dua perkalian untuk setiap *2-point butterfly*) sejumlah *v stage* ($N = 2^v$), sehingga komputasi FFT adalah $O(N\log_2 N)$.

Agar perbedaan laju komputasi DFT dan FFT dapat terlihat dengan jelas, kedua komputasi ini akan dibandingkan dengan nilai *N* yang cukup besar yaitu $N = 1024$. Komputasi DFT untuk $N = 1024$ adalah $1024^2 = 1048576$. Komputasi FFT untuk

N = 1024 adalah $1024x\log_2(1024) = 1024x10 = 10240$. Dapat disimpulkan bahwa laju komputasi FFT 102.4 kali lebih cepat daripada DFT. Untuk nilai *N* yang lebih besar, perbedaan laju komputasi akan semakin besar secara signifikan.

2.7 Kompleksitas Algoritma dan Notasi *Big-O*

Sebuah algoritma tidak hanya dilihat dari apakah algoritma tersebut dapat memecahkan masalah atau tidak, tetapi juga dari seberapa efisien algoritma tersebut melakukannya. Efisiensi suatu algoritma dapat diukur dengan menggunakan kompleksitas algoritma yaitu:

- Kompleksitas waktu, diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan suatu algoritma sebagai fungsi dari ukuran masukan (*N*).
- Kompleksitas ruang, diukur dari memori yang digunakan oleh struktur data yang ada di dalam suatu algoritma sebagai fungsi dari ukuran masukan (*N*).

Salah satu hal yang paling penting dalam menghitung kompleksitas komputasi algoritma adalah pertumbuhan dari fungsi kompleksitas dengan meningkatnya besar input (*N*). Kompleksitas algoritma terbagi dalam tiga buah ukuran, yaitu *best case* (Ω), *average case* (Θ), dan *worst case* (O) atau disebut juga dengan notasi *big-O*. *Worst case* atau notasi *big-O* merupakan ukuran sebuah kompleksitas algoritma dimana algoritma tersebut berjalan pada kondisi terburuk.

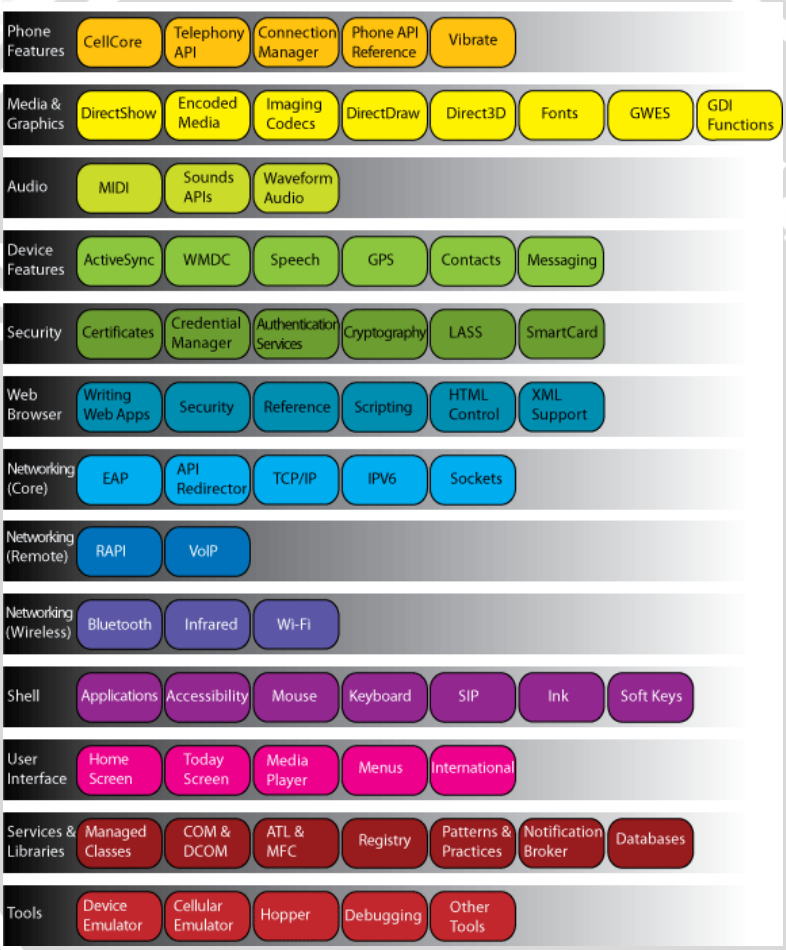
Notasi *big-O* mendefinisikan batas atas (*upper boundary*) dari pertumbuhan suatu fungsi *f(n)* untuk *n* besar. Batas ini diberikan oleh fungsi *g(n)* yang jauh lebih sederhana daripada *f(n)*.

$$O(g(n)) = \{f(n): terdapat\ nilai\ konstan\ c\ dan\ n_0\ yang\ bernilai\ positif,\ sehingga\ 0 \le f(n) \le c.g(n)\ untuk\ semua\ n \ge n_0\} \quad (2\text{-}19)$$

Notasi *big-O* menyederhanakan suatu fungsi agar dapat diketahui tingkat pertumbuhan kompleksitasnya (*growth rates*). Notasi *big-O* dapat ditulis sebagai pangkat tertinggi (*highest order*) dari suatu fungsi polinomial. Hal ini dikarenakan untuk *input* (*N*) yang besar dalam fungsi polinomial, pangkat tertinggi akan sangat mendominasi.

2.8 *Windows Mobile*

Windows Mobile adalah sebuah sistem operasi *mobile* yang dikembangkan oleh Microsoft, dan digunakan untuk berbagai *smartphone* atau *mobile phone*. *Windows Mobile* memiliki fitur-fitur berupa sekumpulan aplikasi yang dikembangkan dari *Microsoft Windows API*. Sistem operasi *Windows Mobile* didesain menjadi semirip mungkin dengan *Windows* versi *desktop*, baik secara fitur dan tampilan. Akan tetapi, fitur yang dimiliki terbatas dan tidak mencakup semua fitur-fitur yang dapat dijalankan pada *Windows* versi *desktop*.



Gambar 2.27 Fitur-fitur pada *Windows Mobile*

(**Sumber :** http://msdn.microsoft.com)

Salah satu keunggulan dari *Windows Mobile* adalah kemudahan penggunaan, karena sebagian besar teknologi informatika berbasis *Windows* sehingga *Windows Mobile* tidak akan terasa asing. Selain itu *Windows Mobile* memiliki kecocokan dengan berbagai jenis *software* yang memang banyak menggunakan *platform Windows*.

2.8.1 *Waveform Audio*

Waveform Audio API merupakan salah satu layanan yang disediakan oleh *Windows Mobile* untuk pengembangan aplikasi. *Waveform Audio API* menyediakan kontrol aplikasi yang berhubungan dengan perangkat *input*/*output* untuk *waveform audio* atau biasa disebut dengan format WAV. Layanan ini mendukung pengelolaan data gelombang audio pada saat perekaman dan pemutaran suara. Implementasi *Waveform Audio API* membutuhkan perangkat keras yang mendukung spesifikasi audio tertentu.

2.8.2 *Graphics Device Interface* (GDI)

GDI merupakan salah satu layanan yang disediakan oleh *Windows Mobile* untuk pengembangan aplikasi. GDI digunakan untuk mengontrol tampilan teks dan grafis. GDI menyediakan komponen-komponen yang dapat digunakan untuk menghasilkan *output* grafis, baik untuk ditampilkan maupun dicetak. GDI dapat digunakan untuk menggambar garis, kurva, teks dan lain-lain. Tiga objek gambar utama yang disediakan oleh GDI yaitu:

- *Pen* : untuk menggambar garis dan kurva
- *Brush* : untuk mengisi objek dengan warna tertentu
- *Font* : untuk menulis teks

2.9 Bahasa Pemrograman C++

C++ adalah sebuah bahasa pemrograman tingkat menengah yang merupakan perkembangan dari bahasa C, dimana bahasa C++ dikembangkan menjadi bahasa pemrograman yang berorentasi objek. Bahasa C++ merupakan bahasa tingkat menengah karena bahasa ini berada di antara bahasa tingkat tinggi dan rendah. Bahasa tingkat tinggi merupakan bahasa dengan perintah – perintah yang mudah dimengerti karena lebih berorientasi kepada bahasa manusia. Sedangkan bahasa tingkat rendah adalah bahasa yang lebih berorientasi kepada bahasa mesin dengan menggunakan kode biner atau heksadesimal.

Bahasa pemrograman C++ untuk pengembangan aplikasi visual terbagi dalam *manage code* dan *unmanaged (native) code*. *Manage code* adalah kode yang dieksekusi dan dikontrol oleh sebuah lingkungan eksekusi (*framework*) tertentu yang menghubungkan aplikasi dengan sistem operasi. *Native code* adalah kode yang berada di bawah kendali sistem operasi secara langsung dan dikompilasi dalam bahasa mesin.

Aplikasi berbasis *Windows* yang dibangun menggunakan *native code* Visual C++ disebut juga dengan aplikasi *Win32* karena menggunakan *library-library Win32 API*. Kode dari aplikasi *Win32* langsung mengakses API dari sistem operasi *Windows* tanpa menggunakan *library* atau *framework* lain. Keuntungan yang didapat dari aplikasi *Win32* adalah kecepatan dalam eksekusi program, serta ukuran *file* aplikasi yang relatif kecil.

BAB III
METODOLOGI PENELITIAN

Metodologi yang digunakan pada penulisan skripsi ini meliputi langkah-langkah yang akan dilakukan untuk merancang dan membangun aplikasi penganalisis spektrum sinyal digital. Langkah-langkah yang akan dilakukan yaitu:

1. Studi Literatur
2. Perancangan dan Implementasi Sistem
3. Pengujian Sistem
4. Kesimpulan dan Saran

3.1 Studi Literatur

Studi literatur dilakukan untuk mempelajari dan memahami konsep dasar yang digunakan untuk menunjang penulisan skripsi. Studi literatur yang dilakukan meliputi:

1. Teori-teori dasar mengenai pemrosesan sinyal audio digital, analisis spektrum dan *Fast Fourier Transform*.
2. Karakteristik perangkat *mobile phone* yang digunakan sebagai media perangkat lunak.
3. Teknik-teknik dasar pemrograman menggunakan bahasa pemrograman C++ untuk perangkat berbasis sistem operasi *Windows Mobile*.

3.2 Perancangan dan Implementasi Sistem

Pada tahap ini dilakukan perancangan sistem dari aplikasi penganalisis spektrum sinyal digital yang bekerja secara *real-time* pada perangkat *mobile phone*, dengan menggunakan metode FFT. Garis besar perancangan sistem dapat dibagi menjadi dua bagian, yaitu perekaman suara dan pemrosesan data.

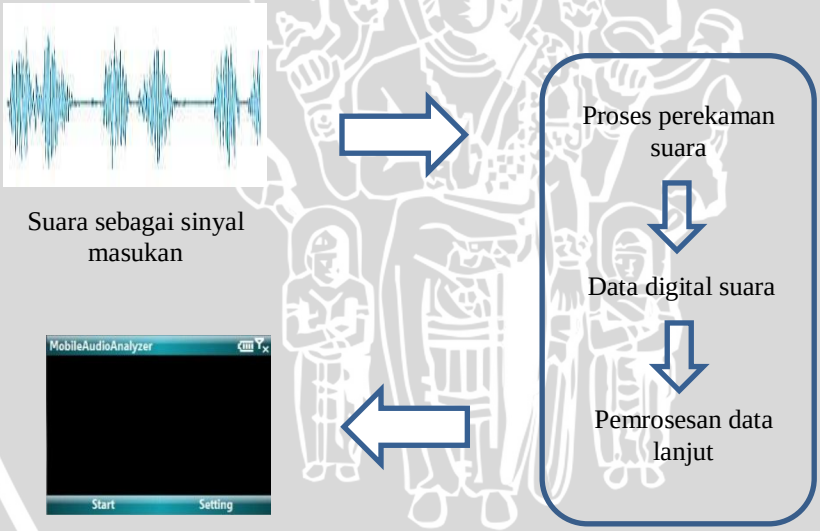
Proses perekaman suara (*recording*) dilakukan untuk mendapatkan *sample* gelombang suara sebagai data *input* dari sistem, dengan media yang digunakan yaitu *microphone*. Data *sample* diperoleh dari hasil pemrosesan sinyal analog ke digital yang dilakukan oleh komponen ADC. Untuk penyimpanan data *sample*, disiapkan sejumlah *buffer* yang diantrikan secara berputar (*circle-queue*).

Pemrosesan data dilakukan setiap terdapat *buffer* yang telah terisi penuh. Pemrosesan data yang dilakukan antara lain:

1. Perhitungan algoritma FFT, dilakukan untuk mentransformasi data *sample* yang berada pada *domain* waktu menjadi *domain* frekuensi.
2. Pembaharuan (*update*) tampilan pada layar, untuk menampilkan gelombang atau spektrum frekuensi dari suara. Untuk menampilkan spektrum frekuensi, diperlukan perhitungan amplitudo dan frekuensi dari hasil perhitungan sebelumnya menggunakan FFT.

Kedua proses utama baik perekaman suara maupun pemrosesan data dilakukan secara paralel, kontinyu dan cepat, sehingga dapat dihasilkan sebuah sistem aplikasi yang bekerja dengan prinsip waktu-nyata (*real-time*). Aplikasi ini akan dijalankan pada perangkat *mobile phone* dengan komponen *microphone* serta ADC (*Analog to Digital Converter*) yang dapat berfungsi dengan baik.

Desain rancangan sistem dari aplikasi penganalisis spektrum sinyal digital dapat digambarkan sebagai berikut.



Gambar 3.1 Desain umum rancangan sistem

(**Sumber :** Perancangan)

Cara kerja dari aplikasi penganalisis spektrum sinyal digital adalah sebagai berikut.

1. Sistem akan mulai dijalankan ketika *user* telah memilih hasil tampilan yang diinginkan (*domain* waktu atau *domain* frekuensi).
2. Sistem menjalankan fungsi *recording*, data *sample* disimpan pada antrian *buffer*.
3. Jika terdapat *buffer* yang terisi penuh, sistem menjalankan fungsi algoritma FFT dan *update* tampilan pada layar.
4. Untuk tampilan gelombang sinusoida sinyal, fungsi algoritma FFT tidak dijalankan. Data *input* ditampilkan pada layar secara langsung.
5. Untuk tampilan spektrum frekuensi sinyal, data yang ditampilkan pada layar diambil dari data hasil perhitungan FFT.
6. Perekaman dan pemrosesan data akan dilakukan secara terus menerus, hingga *user* menghentikan fungsi *recording* (*stop*) atau *user* keluar dari aplikasi (*exit*).

Tahap selanjutnya adalah implementasi sistem. Pada tahap ini dilakukan transformasi hasil perancangan sistem yang telah dibuat ke dalam kode *(coding)* sesuai dengan bahasa pemrograman yang digunakan.

3.3 Pengujian Sistem

Pengujian sistem dilakukan untuk mengetahui apakah sistem bekerja dengan baik dan sesuai dengan perancangan. Pengujian yang dilakukan adalah sebagai berikut:

1. Pengujian hasil tampilan

   Pengujian hasil tampilan dilakukan untuk mengetahui respon aplikasi terhadap sinyal masukan berupa suara dengan beberapa kondisi tertentu.
2. Perhitungan kompleksitas algoritma

   Perhitungan kompleksitas algoritma dilakukan untuk mengetahui besar waktu yang dibutuhkan untuk menjalankan sistem. Kompleksitas algoritma adalah sebuah model abstrak pengukuran waktu atau ruang yang tidak terpengaruh oleh mesin maupun *compiler* yang digunakan.
3. Analisis sistem

   Menjelaskan analisis sistem dari hasil pengujian.

Tahapan pengujian sistem akan dilakukan pada perangkat *mobile* (*mobile phone*) dengan spesifikasi sebagai berikut:

- Sistem Operasi : Microsoft Windows Mobile 6.0 Standard
- Prosesor : ARM1136OMAP2420
- RAM (Memori) : 96 MB
- Resolusi layar : 320 x 240 pixel

3.4 Pengambilan Kesimpulan dan Saran

Pada tahap ini diambil kesimpulan dari hasil pengujian dan analisis terhadap aplikasi penganalisis spektrum sinyal digital. Tahap selanjutnya adalah membuat saran untuk perbaikan dan penyempurnaaan kekurangan-kekurangan yang ada, sehingga dapat dilakukan pengembangan selanjutnya.

BAB IV
PERANCANGAN DAN IMPLEMENTASI

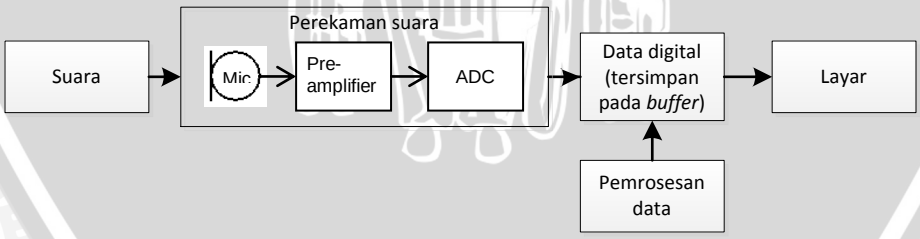
Perancangan dan implementasi dari aplikasi penganalisis spektrum sinyal digital yang bekerja secara *real-time* pada perangkat *mobile phone* dengan menggunakan metode FFT, dilakukan dalam beberapa tahap meliputi perekaman suara (*recording*), mekanisme penyimpanan dan pemrosesan data, algoritma FFT, dan pembaharuan (*update*) tampilan pada layar. Pembuatan secara bertahap ini dimaksudkan untuk memudahkan analisis aplikasi pada setiap bagian maupun aplikasi secara keseluruhan.

4.1 Perancangan Secara Umum

Perancangan aplikasi secara global merupakan tahap awal dan digunakan sebagai acuan dalam perancangan aplikasi yang akan dibuat. Perancangan aplikasi diawali dengan pendefinisian kegiatan *user* atau pengguna dalam menggunakan aplikasi penganalisis spektrum sinyal digital. Perancangan secara umum meliputi blok diagram sistem, cara kerja aplikasi dan konfigurasi perangkat keras.

4.1.1 Blok Diagram Sistem

Aplikasi penganalisis spektrum sinyal digital menggunakan beberapa komponen yang dapat digambarkan secara umum sebagai berikut.



Gambar 4.1 Diagram blok sistem

(**Sumber :** Perancangan)

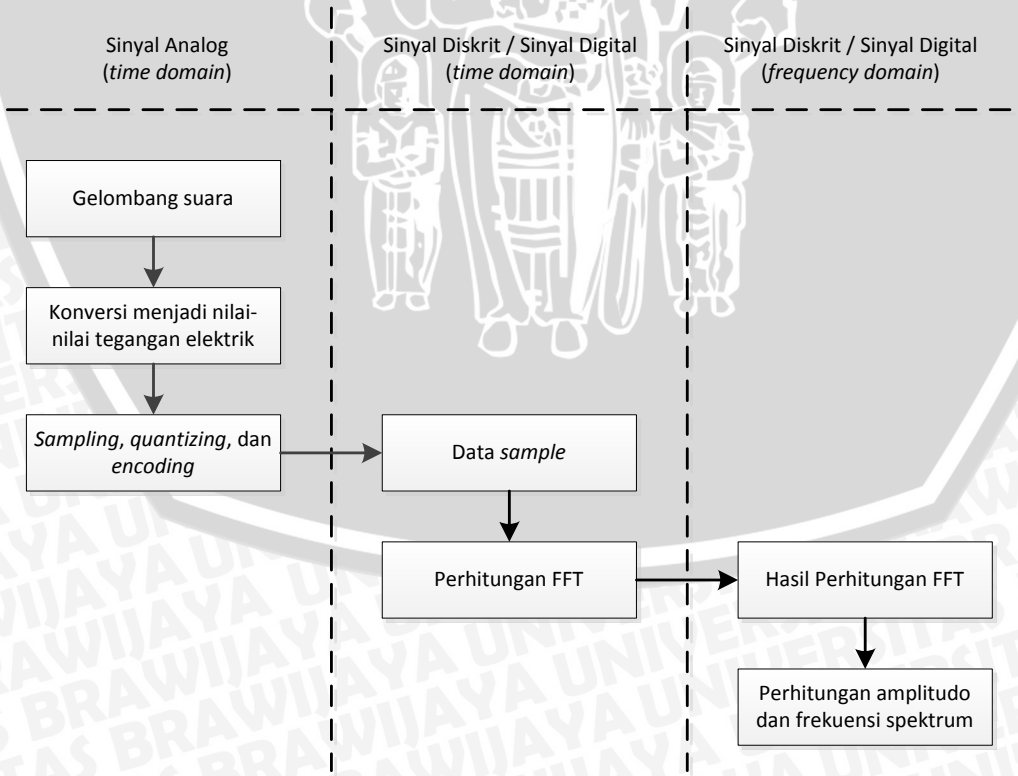
Fungsi masing-masing bagian dalam diagram blok sistem di atas adalah sebagai berikut.

1. Suara merupakan sinyal *input* yang digunakan sistem.
2. *Microphone* digunakan sebagai media *recording*, serta berfungsi untuk mengubah gelombang suara ke dalam nilai-nilai tegangan elektrik.

3. *Pre-amplifier* digunakan untuk menguatkan nilai-nilai tegangan yang didapatkan dari *microphone*.
4. ADC atau *Analog to Digital Converter* merupakan komponen yang bertugas melakukan konversi sinyal analog menjadi sinyal digital. Proses yang dilakukan yaitu *sampling*, *quantizing* dan *encoding*.
5. Data *sample* hasil *recording* adalah data digital dari suara, yang disimpan pada *buffer*. *Buffer* digunakan sebagai tempat penampungan data sementara. Data *sample* inilah yang akan diproses oleh sistem.
6. Layar (*mobile phone*) digunakan sistem untuk menampilkan gelombang atau spektrum frekuensi dari suara.

Komponen-komponen perekam suara terdapat pada *mobile phone* dan merupakan komponen utama yang digunakan dalam proses *recording*. Sedangkan *buffer* dialokasikan oleh sistem pada memori.

Suara akan melalui tiga fase bentuk sinyal agar dapat menampilkan spektrum frekuensinya. Bentuk sinyal ini antara lain: sinyal analog *domain* waktu; sinyal digital *domain* waktu dan sinyal digital *domain* frekuensi. Perubahan sinyal digital *domain* waktu menjadi *domain* frekuensi dilakukan dengan menggunakan metode FFT. Blok diagram perubahan bentuk sinyal *input* dapat digambarkan sebagai berikut.



Gambar 4.2 Blok diagram perubahan bentuk sinyal *input* untuk menampilkan spektrum frekuensi

(**Sumber :** Perancangan)

4.1.2 Cara Kerja Aplikasi

Aplikasi penganalisis spektrum sinyal digital memiliki dua proses utama yang berjalan terpisah, yaitu perekaman suara (*recording*) dan pemrosesan data. Kedua proses ini tidak saling berurutan, artinya proses satu tidak menunggu proses yang lain untuk dapat dijalankan. Hal ini menghasilkan dua proses yang dapat berjalan secara hampir bersamaan, atau disebut sebagai *parallel processing*.

Proses perekaman suara (*recording*) diawali dengan membuka koneksi sistem dengan *microphone*. *Microphone* bekerja dengan mendeteksi adanya perubahan nilai tekanan udara yang dihasilkan sumber suara, yang kemudian diubah ke dalam nilai tegangan elektrik. Nilai tegangan elektrik ini selanjutnya dikonversi menjadi data digital oleh ADC.

Proses konversi ADC dilakukan dengan mencuplik (*sampling*) gelombang suara berupa nilai-nilai tegangan terhadap interval waktu tertentu. Nilai tegangan setiap *sample* kemudian dibandingkan dengan level kuantisasi (*quantizing*). Hasil akhir dari proses konversi adalah data *sample* yang berupa serangkaian data digital.

Data s*ample* yang diperoleh dari proses *recording* akan disimpan sementara ke dalam sejumlah *buffer*. *Buffer-buffer* ini diantrikan secara berputar (*circle-queue*), sehingga penyimpanan data dapat terus dijalankan. Setiap *buffer* terisi penuh, data yang tersimpan pada *buffer* akan diproses. Kemudian *buffer* akan diantrikan kembali untuk persiapan penyimpanan data *sample* berikutnya.

Pemrosesan data pada *buffer* tidak mempengaruhi jalannya proses *recording*. Ketika *buffer* penuh, sistem akan melakukan pemrosesan data sekaligus melanjutkan proses *recording* pada *buffer* berikutnya. Hal ini dilakukan agar tidak ada informasi yang terlewat pada proses *recording*.

Pemrosesan data dilakukan sesuai pilihan tampilan yang diinginkan *user*. Jika user memilih tampilan gelombang sinusoida sinyal, maka data *sample* akan ditampilkan pada layar secara langsung. Jika *user* memilih tampilan spektrum frekuensi sinyal, maka data *sample* akan diproses terlebih dahulu menggunakan algoritma FFT. Setelah perhitungan amplitudo dan frekuensi dilakukan, spektrum frekuensi akan ditampilkan pada layar.

Aplikasi penganalisis spektrum sinyal digital ini akan terus menjalankan proses *recording*, hingga *user* menghentikan proses (*stop*) atau *user* keluar dari aplikasi (*exit*). Ketika *user* ingin mengganti tampilan yang diinginkan, proses *recording* juga akan di hentikan secara otomatis.

4.1.3 Konfigurasi Perangkat Keras

Aplikasi penganalisis spektrum sinyal digital dapat bekerja dengan baik bila sinyal *input* yang akan dianalisis dapat terekam dengan *noise* yang sekecil-kecilnya. Untuk itu diperlukan penanganan perangkat *mobile* (*mobile phone*) yang tepat pada lapangan tempat sistem diimplementasikan. Penanganan yang diperlukan adalah sebagai berikut.

1. Memastikan tidak ada aplikasi lain pada *mobile phone* yang masih berjalan.
2. Memastikan sumber suara sinyal *input* berada pada jarak yang cukup dekat dengan *mobile phone*.
3. Menjauhkan *mobile phone* dari sumber suara lain agar suara-suara yang tidak diinginkan (*noise*) tidak ikut terekam.

Gambar 4.3 Sebaran suara yang dapat memperbesar *noise* sinyal *input*
(**Sumber :** Perancangan)

4.2 Perancangan Perangkat Lunak

Perangkat lunak dibangun dengan menggunakan bahasa pemrograman C++. Perancangan sistem dilakukan dengan spesifikasi sebagai berikut.

1. Melakukan perekaman suara (*recording*)
2. Mengatur mekanisme penyimpanan dan pemrosesan data
3. Melakukan perhitungan algoritma FFT
4. Melakukan pembaharuan (*update*) tampilan pada layar

4.2.1 Perekaman Suara (*Recording*)

Perekaman suara dilakukan untuk mendapatkan data *sample* dari sinyal *input*. Sebelum perekaman suara dimulai, terlebih dahulu dilakukan persiapan-persiapan perekaman suara meliputi inisialisasi format perekaman suara, inisialisasi *buffer* dan koneksi *microphone*.

Spesifikasi format yang digunakan untuk perekaman suara antara lain:

1. *Audio format*

   Format audio yang digunakan adalah format PCM.

2. *Channel*

   Jumlah *channel* yang digunakan yaitu satu *channel (mono*).

3. *Sampling rate*

   Besar *sampling rate* yang digunakan adalah 8000 Hz, jadi pencuplikan akan dilakukan sebanyak 8000 *samples* per detik.

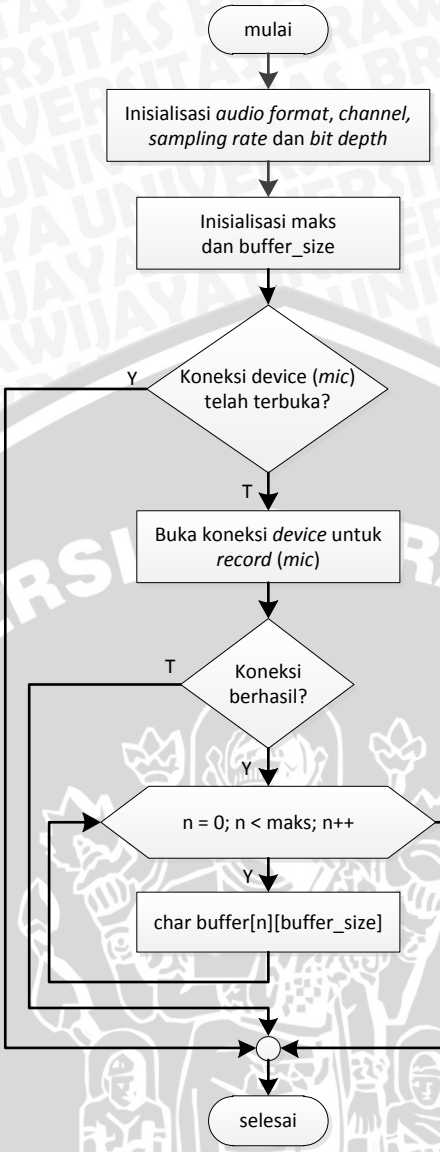
4. *Bit depth*,

   Besar *bit depth* yang digunakan adalah 8 bit, jadi resolusi atau jumlah nilai level kuantisasi yang memungkinkan adalah 256 (2^8) level. Rentang level yang dimiliki yaitu 0 s/d 255 atau -128 s/d 127.

Untuk keperluan penyimpanan data *sample*, akan disiapkan 16 buah *buffer* dengan masing-masing *buffer* dapat menampung 256 data *sample*. Ukuran penampungan *buffer* sebesar 256 data *sample* diperoleh dari beberapa pertimbangan yaitu:

1. Data *sample* akan ditampilkan pada layar sesuai jumlah data atau titik yang diproses, jadi jumlah data tidak boleh melebihi lebar layar yaitu 320 pixel.
2. Syarat algoritma FFT adalah data *input* yang akan dihitung harus sejumlah perpangkatan dua. Jumlah yang paling mendekati dengan mempertimbangkan lebar layar adalah 256 (2^8). Sehingga data *sample* akan diproses dan ditampilkan setiap 256 titik dan algoritma FFT yang digunakan adalah *256-Point* FFT.

Diagram alir untuk persiapan perekaman suara ditunjukkan pada pada Gambar 4.4.



Gambar 4.4 Diagram alir persiapan perekaman suara

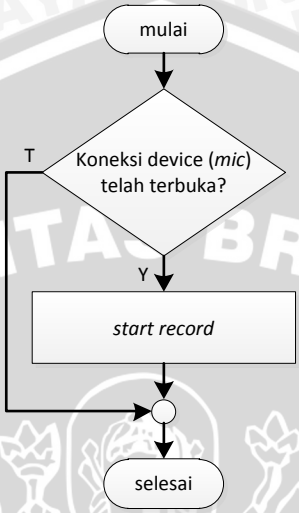
(**Sumber :** Perancangan)

Penjelasan diagram alir persiapan perekaman suara adalah sebagai berikut.

1. Mendefinisikan spesifikasi format perekaman suara yaitu *sampling rate*, *bit depth* dan *channel* serta buffer_size sebagai ukuran dari *buffer*.
2. Mendefinisikan jumlah *buffer* (maks) dan ukuran buffer (buffer_size).
3. Melakukan pengecekan apakah koneksi dengan *microphone* telah terbuka. Jika koneksi telah terbuka maka keluar.
4. Jika koneksi belum terbuka maka membuka koneksi dengan *microphone*.
5. Melakukan pengecekan apakah koneksi telah berhasil atau gagal. Jika koneksi gagal maka keluar.

6. Jika koneksi berhasil maka dilakukan deklarasi buffer sebanyak jumlah *buffer* (maks).

Perekaman suara dapat dijalankan setelah persiapan-persiapan selesai dilakukan. Diagram alir perekaman suara ditunjukkan pada Gambar 4.5.



Gambar 4.5 Diagram alir perekaman suara

(**Sumber :** Perancangan)

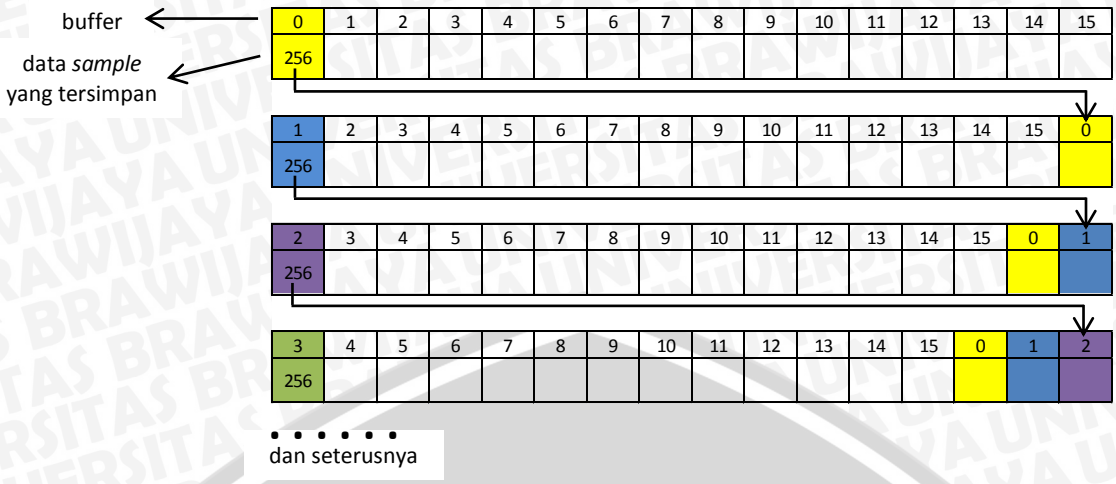
Penjelasan diagram alir perekaman suara adalah sebagai berikut.

1. Melakukan pengecekan apakah koneksi dengan *microphone* telah terbuka. Jika koneksi belum terbuka maka keluar.
2. Jika koneksi telah terbuka maka perekaman suara dimulai.

4.2.2 Mekanisme Penyimpanan dan Pemrosesan Data

Penyimpanan data *sample* pada *buffer* akan terus dilakukan selama proses perekaman suara berlangsung. Data disimpan pada *buffer* sesuai urutan antrian *buffer* tersebut.

Penyimpanan data *sample* dimulai dari *buffer* di awal antrian yaitu buffer0. Setelah buffer0 penuh berisi 256 data, penyimpanan dilanjutkan pada *buffer* berikutnya (buffer1), sedangkan buffer0 diantrikan kembali di akhir antrian (setelah buffer15). Begitu seterusnya hingga buffer15 penuh, maka penyimpanan data kembali dilakukan pada buffer0.

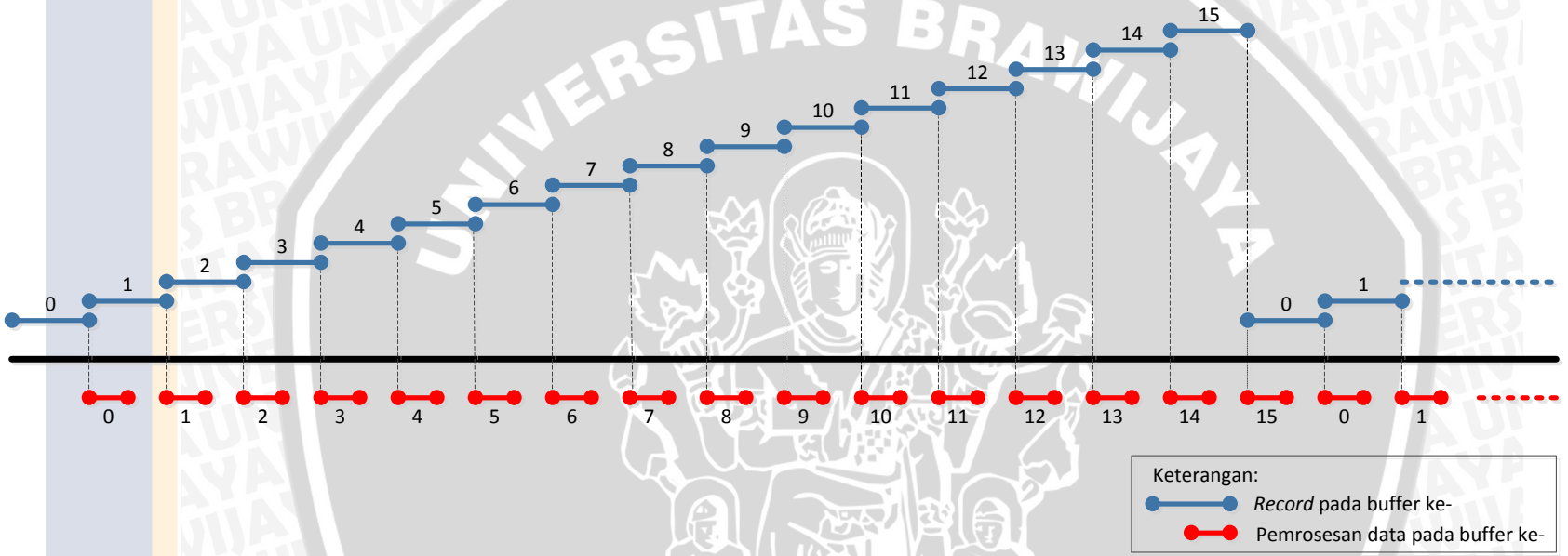


Gambar 4.6 Mekanisme penyimpanan data pada *buffer*
(**Sumber :** Perancangan)

Ketika sebuah *buffer* penuh, sistem akan menjalankan dua proses yang berbeda yang disebut *parallel processing*. Proses yang pertama adalah melanjutkan penyimpanan data, sedangkan proses yang kedua adalah pemrosesan data *sample* yang tersimpan pada *buffer*. Pemrosesan data meliputi perhitungan algoritma FFT dan *update* tampilan pada layar. Ilustrasi hubungan antara penyimpanan data dengan pemrosesan data ditunjukkan pada Gambar 4.7.

Dengan *sampling rate* sebesar 8000 Hz dan *buffer* mampu menampung 256 data *sample*, maka dalam satu detik akan dilakukan 32 kali penyimpanan data pada *buffer*. Dengan menggunakan 16 *buffer* berarti masing-masing *buffer* akan diisi kembali setiap ±0.5 detik. Pemrosesan data akan dilakukan setiap ±1/32 detik (0.03125 detik). Jadi dalam kurun waktu kurang lebih satu detik terdapat 32 gambar yang akan ditampilkan satu-persatu pada layar.

Pemrosesan data pada sistem membutuhkan nilai skala pengukuran penuh dimana terdapat nilai positif dan negatif. Dengan *bit depth* sebesar 8 bit dan rentang level kuantisasi yaitu 0 – 255, maka nilai data *sample* yang tersimpan berkisar antara 0 s/d 255. Rentang nilai level 0 – 255 setara dengan -128 – 127. Sehingga untuk mendapatkan nilai skala pengukuran penuh, nilai data *sample* yang tersimpan terlebih dahulu akan dikurangi sebesar 128.



Gambar 4.7 Ilustrasi hubungan antara penyimpanan data dengan pemrosesan data

(**Sumber :** Perancangan)

4.2.3 Algoritma FFT

Perhitungan algoritma FFT dilakukan untuk mengubah data *sample* sinyal dalam *domain* waktu menjadi *domain* frekuensi. Sebelum dilakukan perhitungan FFT, terlebih dahulu dilakukan persiapan-persiapan meliputi inisialisasi *N-Point* FFT dan inisialisasi susunan *bit-reversed*.

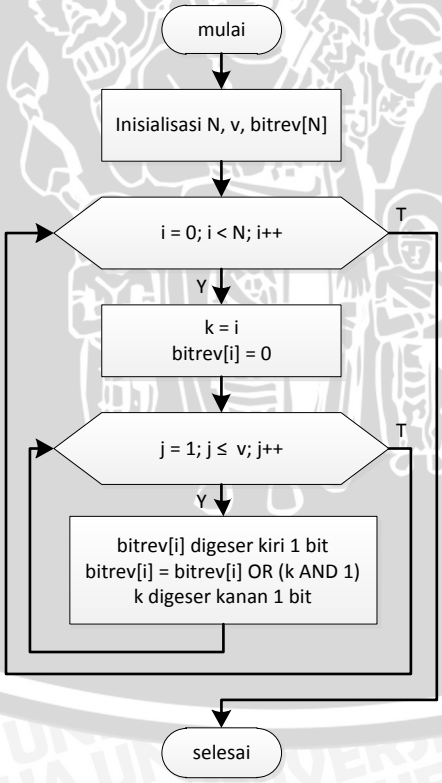
Data *sample* yang diproses yaitu sejumlah 256 titik, jadi perhitungan FFT yang dilakukan adalah *256-Point* FFT. Jumlah *stage/v* dapat dihitung menggunakan persamaan

$$N = 2^v \tag{4-1}$$

Untuk $N = 256 = 2^8$ maka jumlah *stage* = 8.

Susunan *bit-reversed* digunakan sebagai urutan data *sample* yang akan diproses. Data *sample* yang disusun secara *bit-reversed* akan mengembalikan susunan hasil akhir perhitungan FFT pada urutan yang normal.

Diagram alir untuk persiapan perhitungan FFT ditunjukkan pada Gambar 4.8.



Gambar 4.8 Diagram alir persiapan perhitungan FFT

(**Sumber :** Perancangan)

Penjelasan diagram alir persiapan perhitungan FFT adalah sebagai berikut.

1. Mendefinisikan *N-Point*, v-*stage*, serta bitrev sepanjang *N* yang digunakan untuk penyimpanan nilai *bit-reversed*.
2. Melakukan perhitungan *bit-reversed* untuk setiap nilai i.
3. Perhitungan *bit-reversed* diawali dengan mendefinisikan nilai k sesuai nilai i, sedangkan bitrev[i] bernilai 0.
4. Kemudian dilakukan pergeseran dan perbandingan setiap bit nilai i. Jumlah bit nilai i dengan jangkauan nilai antara 0 s/d 255 adalah 8 bit.
5. Pertama-tama nilai bitrev[i] digeser ke kiri sebanyak 1 bit. Kemudian dilakukan perkalian bit (AND) antara k dengan 1 untuk mendapatkan nilai LSB. Hasil perkalian selanjutnya dijumlahkan (OR) dengan nilai bitrev[i]. Setelah itu k digeser ke kanan sebanyak 1 bit.
6. Langkah (5) dilakukan sebanyak 8 kali. Hasil perhitungan *bit-reversed* akan tersimpan pada bitrev[i].

Contoh perhitungan *bit-reversed* ditunjukkan pada Tabel 4.1 dengan nilai i = 00010110 (22). Hasil akhir menunjukkan nilai bitrev = 01101000 (104), yang merupakan hasil pembalikan bit dari nilai i.

Tabel 4.1 Perhitungan *bit-reversed* untuk nilai i = 00010110

step	k	bitrev	bitrev digeser kiri 1 bit	bitrev OR (k AND 1)	k digeser kanan 1 bit
0	00010110	00000000	-	-	-
1	00010110	00000000	00000000	00000000	00001011
2	00001011	0000000	00000000	00000001	00000101
3	00000101	00000001	00000010	00000011	00000010
4	00000010	00000011	00000110	00000110	00000001
5	00000001	00000110	00001100	00001101	00000000
6	00000000	00001101	00011010	00011010	00000000
7	00000000	00011010	00110100	00110100	00000000
8	00000000	00110100	01101000	01101000	00000000

(**Sumber :** Perancangan)

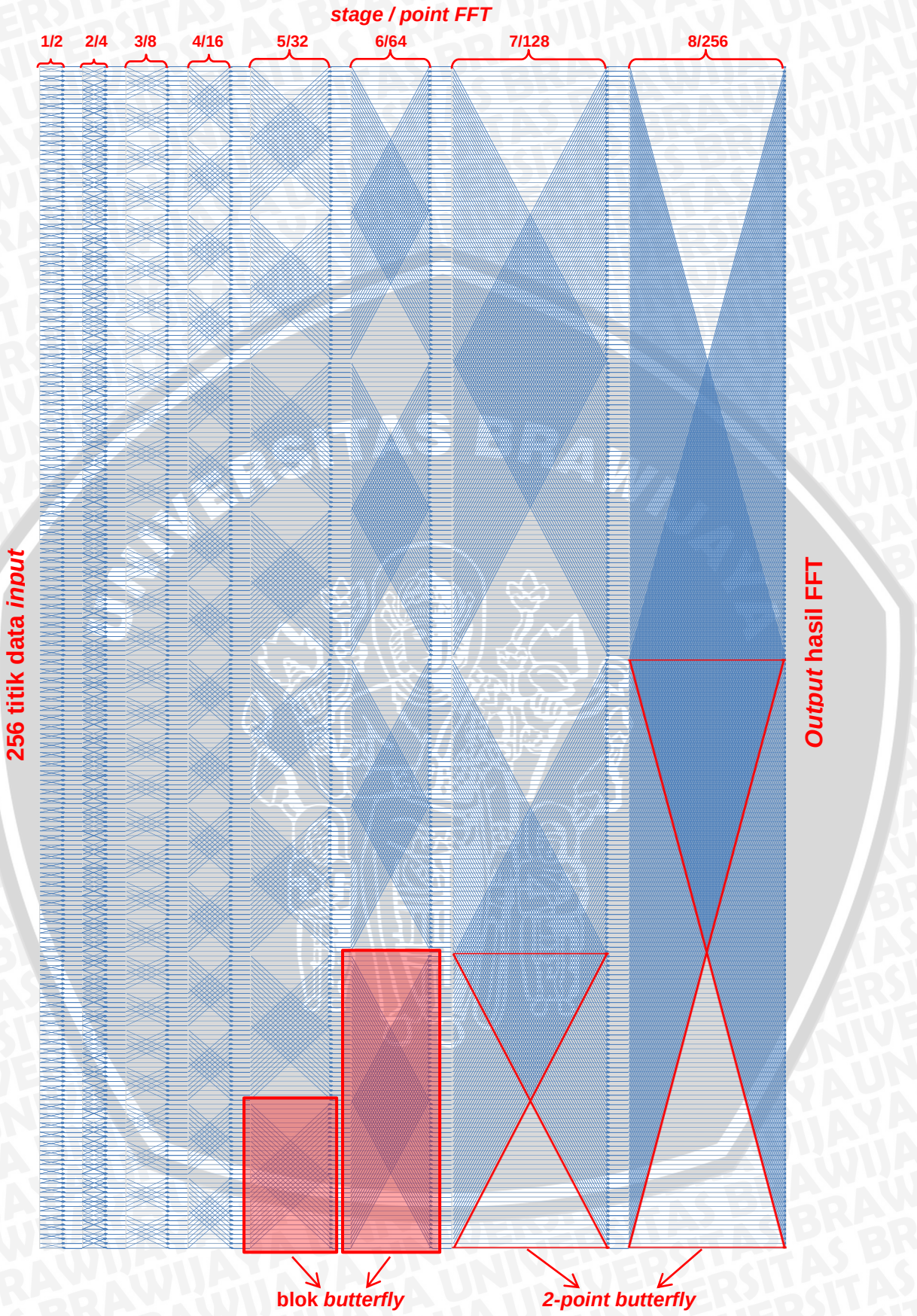
Hasil perhitungan *bit-reversed* sebanyak 256 titik ditunjukkan pada Tabel 4.2.

Tabel 4.2 Hasil perhitungan *bit-reversed* 256 titik

i	*bit-reversed* (i)	i	*bit-reversed* (i)	i	*bit-reversed* (i)	i	*bit-reversed* (i)	i	*bit-reversed* (i)	i	*bit-reversed* (i)
0	0	43	212	86	106	129	129	172	53	215	235
1	128	44	52	87	234	130	65	173	181	216	27
2	64	45	180	88	26	131	193	174	117	217	155
3	192	46	116	89	154	132	33	175	245	218	91
4	32	47	244	90	90	133	161	176	13	219	219
5	160	48	12	91	218	134	97	177	141	220	59
6	96	49	140	92	58	135	225	178	77	221	187
7	224	50	76	93	186	136	17	179	205	222	123
8	16	51	204	94	122	137	145	180	45	223	251
9	144	52	44	95	250	138	81	181	173	224	7
10	80	53	172	96	6	139	209	182	109	225	135
11	208	54	108	97	134	140	49	183	237	226	71
12	48	55	236	98	70	141	177	184	29	227	199
13	176	56	28	99	198	142	113	185	157	228	39
14	112	57	156	100	38	143	241	186	93	229	167
15	240	58	92	101	166	144	9	187	221	230	103
16	8	59	220	102	102	145	137	188	61	231	231
17	136	60	60	103	230	146	73	189	189	232	23
18	72	61	188	104	22	147	201	190	125	233	151
19	200	62	124	105	150	148	41	191	253	234	87
20	40	63	252	106	86	149	169	192	3	235	215
21	168	64	2	107	214	150	105	193	131	236	55
22	104	65	130	108	54	151	233	194	67	237	183
23	232	66	66	109	182	152	25	195	195	238	119
24	24	67	194	110	118	153	153	196	35	239	247
25	152	68	34	111	246	154	89	197	163	240	15
26	88	69	162	112	14	155	217	198	99	241	143
27	216	70	98	113	142	156	57	199	227	242	79
28	56	71	226	114	78	157	185	200	19	243	207
29	184	72	18	115	206	158	121	201	147	244	47
30	120	73	146	116	46	159	249	202	83	245	175
31	248	74	82	117	174	160	5	203	211	246	111
32	4	75	210	118	110	161	133	204	51	247	239
33	132	76	50	119	238	162	69	205	179	248	31
34	68	77	178	120	30	163	197	206	115	249	159
35	196	78	114	121	158	164	37	207	243	250	95
36	36	79	242	122	94	165	165	208	11	251	223
37	164	80	10	123	222	166	101	209	139	252	63
38	100	81	138	124	62	167	229	210	75	253	191
39	228	82	74	125	190	168	21	211	203	254	127
40	20	83	202	126	126	169	149	212	43	255	255
41	148	84	42	127	254	170	85	213	171	-	-
42	84	85	170	128	1	171	213	214	107	-	-

(**Sumber :** Perancangan)

Setelah persiapan-persiapan selesai dilakukan, perhitungan FFT dapat dijalankan. Dasar perhitungan algoritma FFT adalah diagram *butterfly* dua titik (*2-point butterfly*). Untuk *256-Point* FFT dengan 8 *stage*, algoritma FFT dapat diilustrasikan dengan menggunakan diagram *butterfly* seperti yang ditunjukkan pada Gambar 4.9.



Gambar 4.9 Diagram *butterfly* untuk *256-Point* FFT

(**Sumber :** Perancangan)

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Dari ilustrasi *256-Point* FFT pada Gambar 4.9 diperoleh parameter-parameter yang diperlukan dalam perhitungan algoritma FFT sebagai berikut.

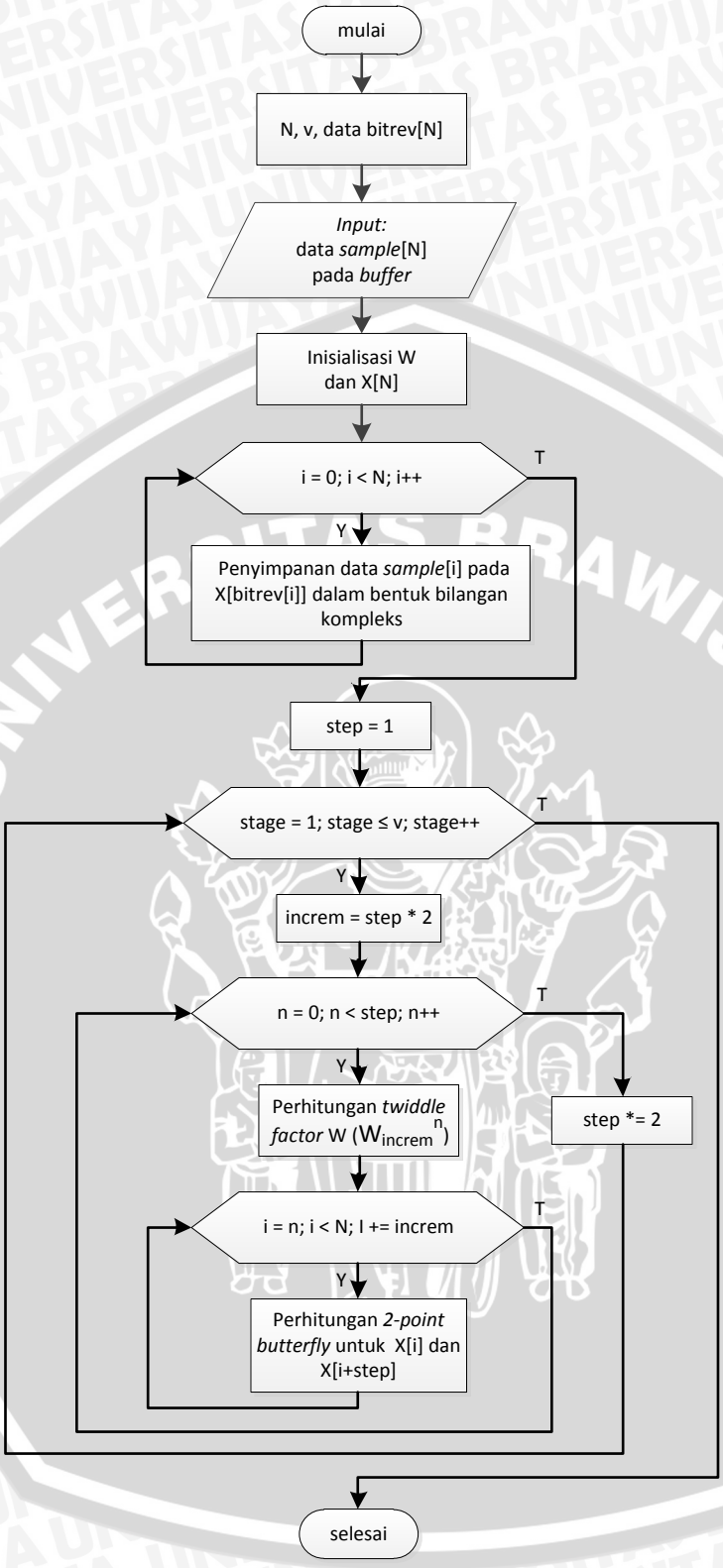
Tabel 4.3 Parameter-parameter perhitungan *256-Point* FFT

Stage	1	2	3	4	5	6	7	8
N-Point FFT	2	4	8	16	32	64	128	256
Jumlah blok *butterfly*	128	64	32	16	8	4	2	1
Jumlah *butterfly* per blok	1	2	4	8	16	32	64	128
Twiddle factor yang digunakan	W_2^{0}	$W_4^{0s/d1}$	$W_8^{0s/d3}$	$W_{16}^{0s/d7}$	$W_{32}^{0s/d15}$	$W_{64}^{0s/d31}$	$W_{128}^{0s/d63}$	$W_{256}^{0s/d127}$

(**Sumber :** Perancangan)

Hasil akhir perhitungan FFT akan berada dalam bentuk bilangan kompleks. Jadi sebelum perhitungan FFT dimulai, data *sample* diubah terlebih dahulu ke dalam bentuk bilangan kompleks dengan nilai *real* = nilai data *sample* dan nilai *imaginer* = 0.

Diagram alir algoritma FFT ditunjukkan pada Gambar 4.10.



Gambar 4.10 Diagram alir algoritma FFT

(**Sumber :** Perancangan)

Penjelasan diagram alir algoritma FFT adalah sebagai berikut.

1. Mengambil variabel N, v, dan bitrev[N] yang telah didefinisikan sebelumnya.
2. Data *input* yaitu data *sample* sejumlah N yang tersimpan pada *buffer*.
3. Mendefinisikan W untuk penyimpanan *twiddle factor*, dan X sepanjang N untuk penyimpanan hasil perhitungan FFT.
4. Menyimpan data *sample*[i] sesuai urutan *bit-reversed* sebagai data *input* algoritma FFT. Nilai data *sample* dikurangi 128 (skala pengukuran penuh) kemudian disimpan pada X[bitrev[i]] dalam bentuk bilangan kompleks. Penyimpanan dilakukan sebanyak jumlah data *sample* N.
5. Mendefinisikan step dengan nilai awal = 1. Variabel step mewakili jumlah *butterfly* per blok, serta jarak antara dua titik *butterfly* yang akan dihitung, pada tiap-tiap *stage*. Setiap perubahan *stage*, variabel step akan dikalikan dua (step = step*2).
6. Pada setiap *stage* terdapat variabel increm dengan nilai increm = step*2. Variabel increm mewakili besar *N-Point* FFT, serta jarak antar blok *butterfly*, pada tiap-tiap *stage*.
7. *Stage* 1 : *2-Point* FFT; step = 1; increm = 2
   - step 0 : perhitungan diagram *butterfly* (*twiddle factor* = W_2^0) untuk 128 blok
8. *Stage* 2 : *4-Point* FFT; step = 2; increm = 4
   - step 0 : perhitungan diagram *butterfly* ke-1 (*twiddle factor* = W_4^0) untuk 64 blok,
   - step 1 : perhitungan diagram *butterfly* ke-2 (*twiddle factor* = W_4^1) untuk 64 blok.
9. Melanjutkan perhitungan untuk *Stage* 3 s/d *Stage* 7.
10. *Stage* 8 : *256-Point* FFT; step = 128; increm = 256
    - step 0 : perhitungan diagram *butterfly* ke-1 (*twiddle factor* = W_{256}^0),
    - step 1 : perhitungan diagram *butterfly* ke-2 (*twiddle factor* = W_{256}^1),

    dan seterusnya hingga,
    - step 127 : perhitungan diagram *butterfly* ke-128 (*twiddle factor* = W_{256}^{127}).
11. Perhitungan *twiddle factor* dilakukan menggunakan persamaan

$$W_{increm}{}^{n} = e^{\frac{-j2\pi n}{increm}} = \cos\frac{2\pi n}{increm} - \mathrm{j}\sin\frac{2\pi n}{increm} \qquad (4\text{-}2)$$

12. Perhitungan *2-point butterfly* dilakukan menggunakan persamaan

$$X[i] = X[i] + (W_{increm}{}^{n})\, X[i + step] \qquad (4\text{-}3)$$

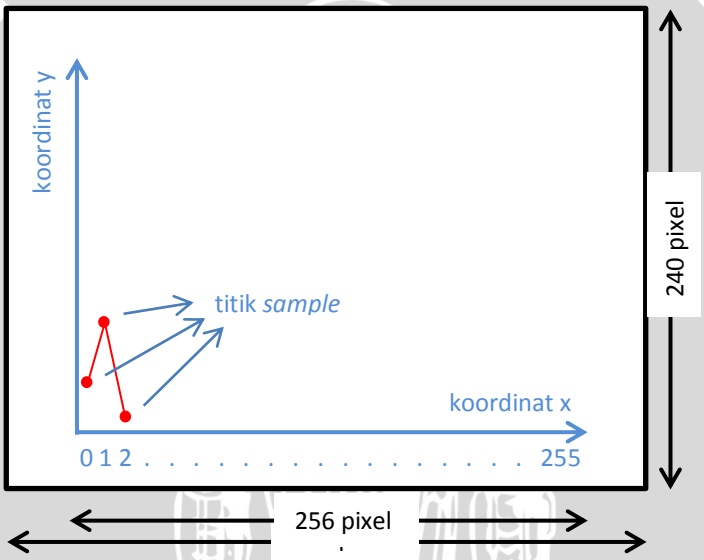
$$X[i + step] = X[i] - (W_{increm}{}^{n})\, X[i + step] \qquad (4\text{-}4)$$

4.2.4 Pembaharuan (*Update*) Tampilan pada Layar

Tampilan pada layar merupakan hasil *output* dari sistem. Gambar yang dihasilkan adalah representasi dari sinyal *input* yang telah diubah menjadi data-data digital. *Update* gambar dilakukan setiap 256 data *sample*, atau setiap pemrosesan data dilakukan.

Data *sample* atau data hasil pemrosesan akan ditampilkan menjadi titik-titik koordinat pada layar. Representasi sinyal *input* didapatkan dari grafik hubungan antar titik-titik koordinat data.

Resolusi layar yang digunakan adalah 320 x 240 pixel. Lebar layar atau koordinat x digunakan untuk rentang titik-titik data *sample* atau data hasil pemrosesan, sedangkan tinggi layar atau koordinat y digunakan untuk skala nilai amplitudo data. Tampilan layar secara umum ditunjukkan pada Gambar 4.11.



Gambar 4.11 Tampilan layar secara umum

(**Sumber :** Perancangan)

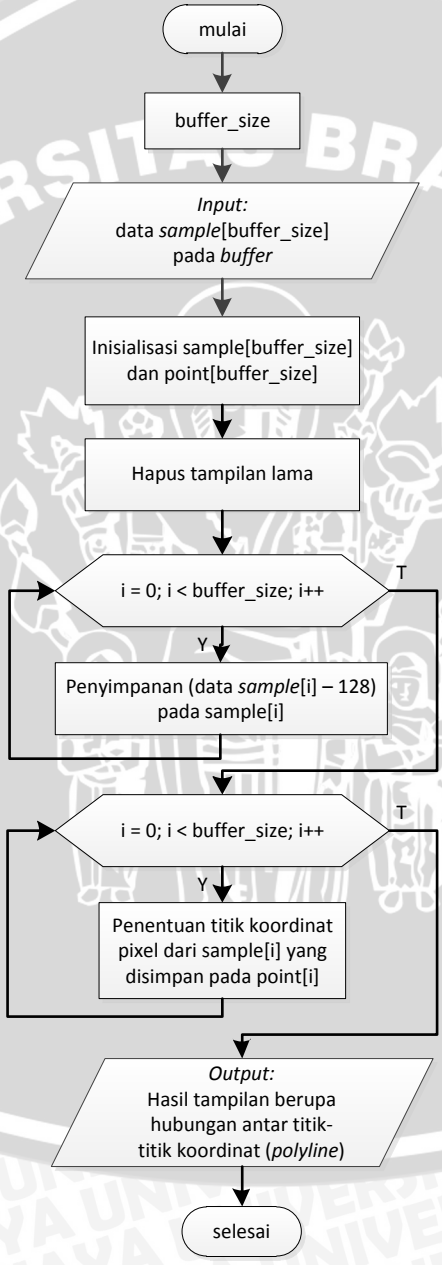
Terdapat tiga macam mode tampilan yang dapat dipilih *user* untuk ditampilkan pada layar yaitu:

1. Tampilan mode grafik sinusoida
2. Tampilan mode grafik spektrum frekuensi skala linear
3. Tampilan mode grafik spektrum frekuensi skala logaritmik

4.2.4.1 Tampilan Mode Grafik Sinusoida

Mode grafik sinusoida digunakan untuk menampilkan representasi sinyal *input* dalam *domain* waktu. Pada mode ini, koordinat x adalah rentang dari 256 titik data *sample*. Sedangkan koordinat y adalah nilai dari titik-titik data *sample* yang berkisar antara -128 s/d 127. Karena jangkauan nilai data untuk koordinat y = 256 melebihi tinggi layar, maka digunakan skala 1:2.

Diagram alir tampilan mode grafik sinusoida ditunjukkan pada Gambar 4.12.



Gambar 4.12 Diagram alir tampilan mode grafik sinusoida

(**Sumber :** Perancangan)

Penjelasan diagram alir tampilan mode grafik sinusoida adalah sebagai berikut.

1. Mengambil variabel buffer_size yang telah didefinisikan sebelumnya.
2. Data *input* yaitu data *sample* sejumlah buffer_size yang tersimpan pada *buffer*.
3. Mendefinisikan sample untuk penyimpanan nilai data *sample* dan point untuk penyimpanan titik-titik koordinat data. Kedua variabel ini didefinisikan sepanjang buffer_size.
4. Menghapus tampilan sebelumnya pada layar.
5. Menyimpan data *sample*[i] ke dalam sample[i]. Nilai data *sample* sebelumnya dikurangi 128 (skala pengukuran penuh). Penyimpanan dilakukan sebanyak buffer_size.
6. Menentukan titik koordinat pixel dari sample[i]. Titik koordinat disesuaikan dengan skala yang digunakan dan letak *zero point* atau titik (0,0). Titik koordinat disimpan pada point[i]. Penyimpanan dilakukan sebanyak buffer_size.
7. Menggabungkan titik-titik koordinat (*polyline*) dan menampilkan hasil grafik pada layar.

4.2.4.2 Tampilan Mode Grafik Spektrum Frekuensi Skala Linear

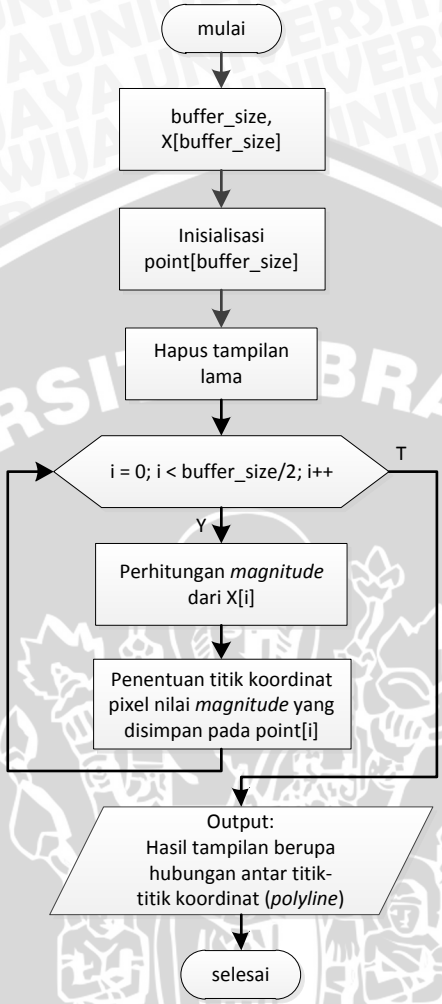
Mode grafik spektrum frekuensi skala linear digunakan untuk menampilkan representasi sinyal *input* dalam *domain* frekuensi. Mode ini diawali dengan pemrosesan data menggunakan perhitungan algoritma FFT.

Koordinat x merupakan nilai frekuensi sinyal. Dengan *sampling rate* sebesar 8000 Hz dan jumlah *N-Point* FFT sebanyak 256, maka resolusi frekuensi sinyal yaitu 8000/256 = 31.25 Hz. Besar frekuensi maksimum adalah setengah dari *sampling rate* yaitu 8000/2 = 4000 Hz. Karena frekuensi maksimum adalah sebesar 4000 Hz, maka hanya 128 titik data yang memenuhi. Spesifikasi koordinat x pada mode ini adalah sebagai berikut:

1. Jumlah titik frekuensi : 128 titik
2. Resolusi frekuensi sinyal : 31.25 Hz
3. Jangkauan frekuensi : 0 – 4000 Hz
4. Jarak antar titik : 2 pixel

Koordinat y merupakan nilai *magnitude* dari hasil perhitungan algoritma FFT. Skala nilai *magnitude* yang digunakan yaitu 1:128.

Diagram alir tampilan mode grafik spektrum frekuensi skala linear ditunjukkan pada Gambar 4.13.



Gambar 4.13 Diagram alir tampilan mode grafik spektrum frekuensi skala linear

(**Sumber :** Perancangan)

Penjelasan diagram alir tampilan mode grafik spektrum frekuensi skala linear adalah sebagai berikut.

1. Mengambil variabel buffer_size dan X[buffer_size] (hasil perhitungan algoritma FFT) yang telah dihitung atau didefinisikan sebelumnya.
2. Mendefinsikan point sepanjang buffer_size untuk penyimpanan titik-titik koordinat data.
3. Menghapus tampilan sebelumnya pada layar.
4. Menghitung nilai *magnitude* dari X[i]. Perhitungan nilai *magnitude* dilakukan menggunakan persamaan

$$|X[i]| = \sqrt{X_{real}[i]^2 + X_{imaginer}[i]^2} \quad (4\text{-}5)$$

5. Menentukan titik koordinat pixel dari hasil perhitungan *magnitude*. Titik koordinat disesuaikan dengan skala yang digunakan dan letak *zero point* atau titik (0,0). Titik koordinat disimpan pada point[i].
6. Perhitungan nilai *magnitude* dan penentuan titik koordinat dilakukan sebanyak buffer_size/2.
7. Menggabungkan titik-titik koordinat (*polyline*) dan menampilkan hasil grafik pada layar.

4.2.4.3 Tampilan Mode Grafik Spektrum Frekuensi Skala Logaritmik

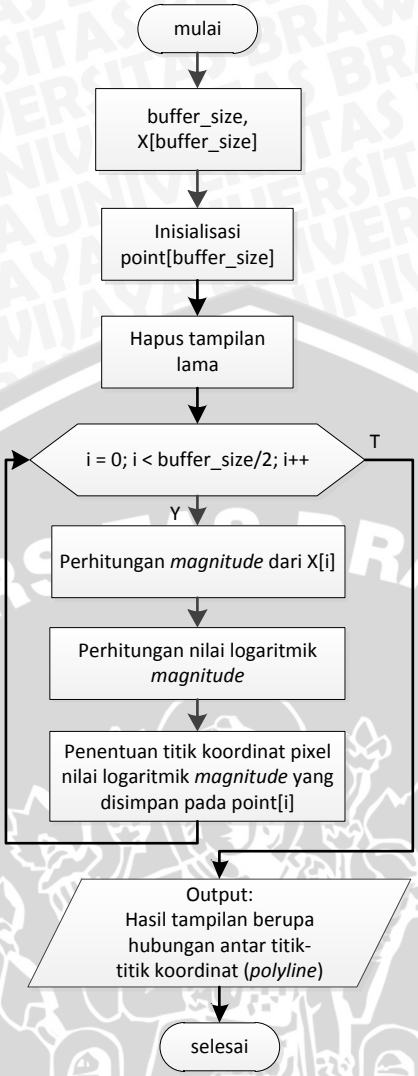
Mode grafik spektrum frekuensi skala logaritmik digunakan untuk menampilkan representasi sinyal *input* dalam *domain* frekuensi. Skala logaritmik yang dimaksud adalah nilai *magnitude* yang ditampilkan dalam satuan desibel (dB).

Mode ini memiliki aliran proses yang hampir sama dengan mode sebelumnya yaitu mode spektrum frekuensi skala linear. Yang membedakan hanyalah perhitungan nilai logaritmik dari *magnitude* sebagai nilai penentu untuk titik koordinat y. Koordinat y berada dalam satuan dB dengan skala 1:1. Perhitungan nilai dB dari *magnitude* dilakukan menggunakan persamaan

$$dB = 20\ log_{10}(|X[i]|) \quad (4\text{-}6)$$

Nilai desibel yang diperoleh menunjukkan bagaimana nilai level tekanan suara didistribusikan sepanjang rentang frekuensi yang dapat dicapai. Hasil keluaran berupa komponen frekuensi sinyal pada koordinat x dan level tekanan suara pada koordinat y. Nilai level tekanan suara yang dihitung dari nilai *magnitude* hasil perhitungan FFT bukan merupakan nilai akhir dari level tekanan suara. Untuk mendapatkan nilai level tekanan suara yang sebenarnya diperlukan perangkat keras tambahan (*microphone* khusus) dan atau perhitungan lebih lanjut.

Diagram alir tamplian mode grafik spektrum frekuensi skala logaritmik ditunjukkan pada Gambar 4.14.

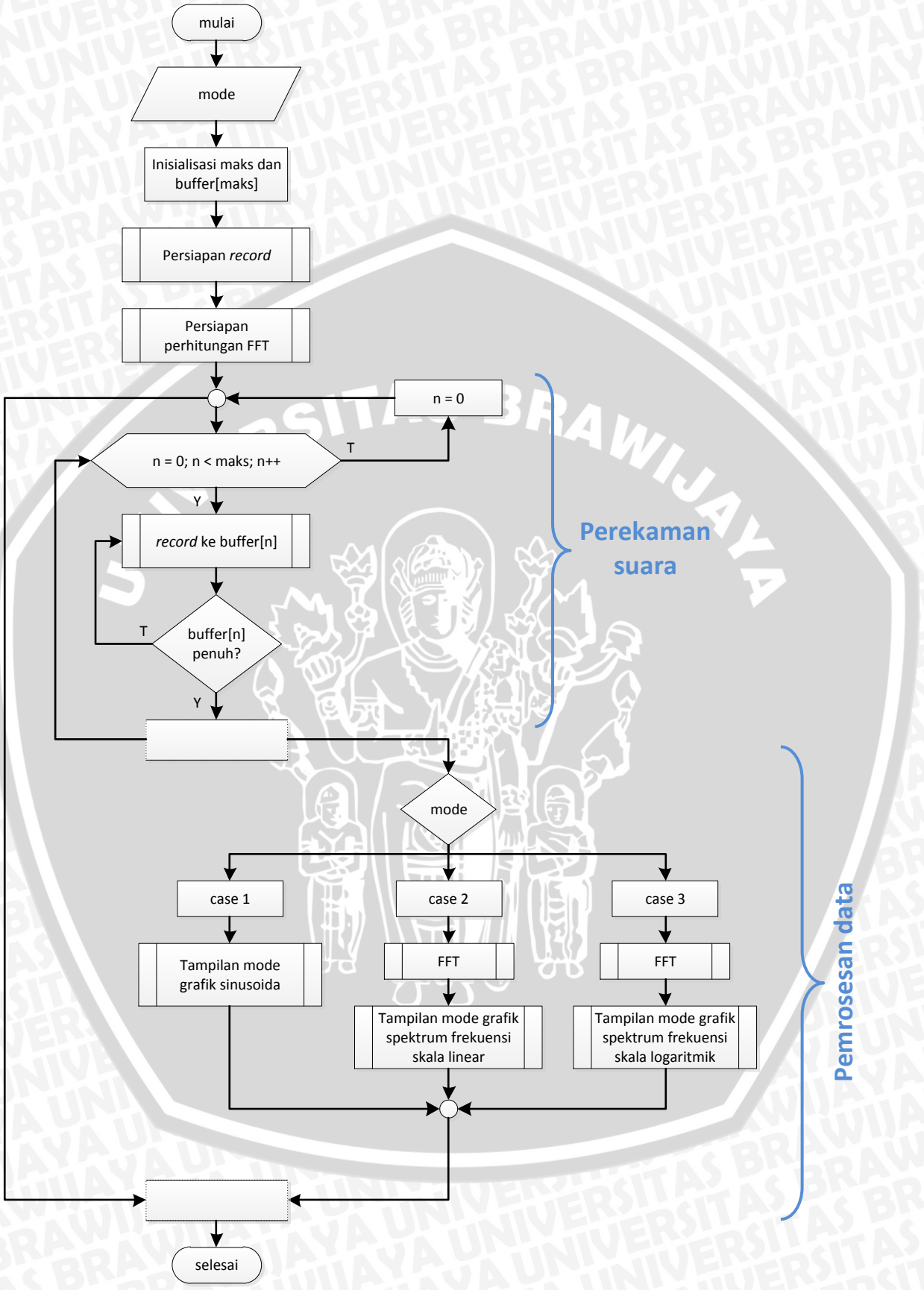


Gambar 4.14 Diagram alir tampilan mode grafik spektrum frekuensi skala logaritmik

(**Sumber:** Perancangan)

4.2.5 Perancangan Aplikasi Keseluruhan

Setelah membuat perancangan fungsi-fungsi, langkah selanjutnya adalah menggabungkan perancangan fungsi-fungsi tersebut menjadi sebuah perancangan aplikasi yang utuh. Diagram alir perancangan aplikasi keseluruhan secara sederhana dapat ditunjukkan pada Gambar 4.15.



Gambar 4.15 Diagram alir perancangan aplikasi keseluruhan

(**Sumber :** Perancangan)

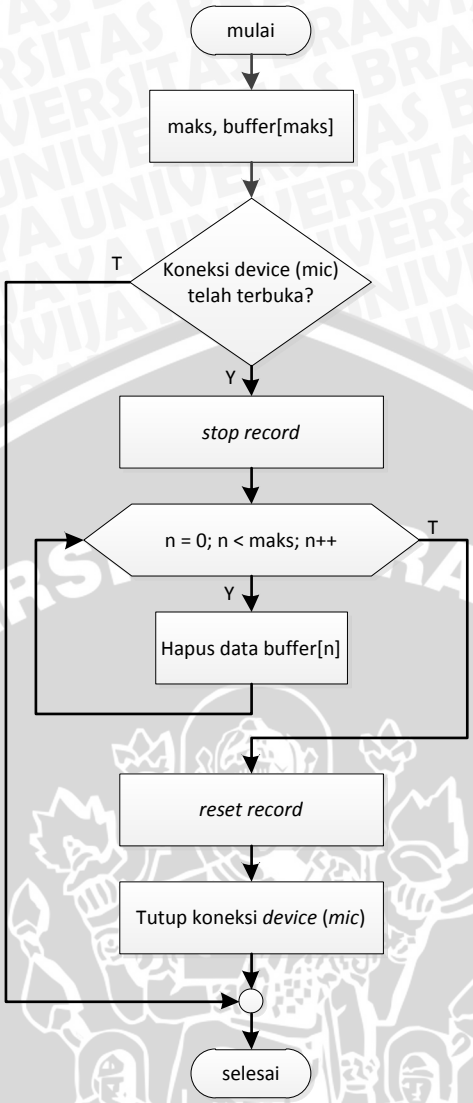
Penjelasan diagram alir perancangan aplikasi secara keseluruhan adalah sebagai berikut.

1. Data *input* yaitu mode yang dipilih oleh *user*.
2. Mendefinisikan jumlah *buffer* (maks) dan buffer sebanyak maks.
3. Menjalankan fungsi persiapan perekaman suara (*recording*).
4. Menjalankan fungsi persiapan perhitungan FFT.
5. Menjalankan fungsi *recording*, data hasil *recording* disimpan pada buffer[n].
6. Melakukan pengecekan apakah buffer[n] telah penuh. Jika belum penuh fungsi *recording* akan terus dilanjutkan.
7. Jika buffer[n] penuh, sistem akan menjalankan dua proses.
8. Proses yang pertama adalah melanjutkan fungsi *recording* pada buffer[n+1]. Proses ini akan terus dilanjutkan hingga buffer[maks-1]. Jika buffer[maks-1] penuh maka proses *recording* dilanjutkan dengan penyimpanan data dimulai kembali dari buffer[0] dan begitu seterusnya.
9. Proses yang kedua adalah pemrosesan data sesuai mode yang dipilih *user*. Jika *user* memilih mode grafik sinusoida maka sistem menjalankan fungsi tampilan mode grafik sinusoida. Jika user memilih mode grafik spektum frekuensi maka sistem akan menjalankan fungsi algoritma FFT, kemudian dilanjutkan dengan fungsi tampilan mode grafik spektrum frekuensi sesuai pilihan *user*.

4.2.6 Menghentikan (*Stop*) Aplikasi

Perancangan aplikasi disesuaikan untuk pemrosesan sinyal digital secara *real-time*, sehingga sistem dijalankan terus-menerus. Oleh karena itu dibutuhkan fungsi untuk menghentikan aplikasi (*stop)*. Fungsi ini akan menghentikan proses *recording* pada sistem, sehingga pemrosesan data juga akan terhenti.

Diagram alir menghentikan aplikasi ditunjukkan pada Gambar 4.16.



Gambar 4.16 Diagram alir menghentikan aplikasi

(**Sumber :** Perancangan)

Penjelasan diagram alir menghentikan aplikasi adalah sebagai berikut.

1. Mengambil variabel maks dan buffer[maks] yang telah didefinisikan sebelumnya.
2. Melakukan pengecekan apakah koneksi dengan *microphone* telah terbuka. Jika koneksi tidak terbuka maka keluar.
3. Jika koneksi terbuka maka proses *recording* dihentikan.
4. Menghapus seluruh data yang ada pada buffer[0] hingga buffer[maks-1].
5. Melakukan *reset* pada proses *recording*. Seluruh antrian *buffer* dihapus dan posisi antrian dikembalikan dari awal.
6. Menutup koneksi dengan *microphone*.

4.3 Implementasi Sistem

Setelah tahap perancangan sistem, tahap selanjutnya adalah implementasi. Tahap implementasi merupakan proses transformasi hasil perancangan perangkat lunak yang telah dibuat ke dalam kode (*coding*) sesuai dengan sintaks dari bahasa pemrograman yang digunakan.

4.3.1 Lingkungan Implementasi

Aplikasi dibangun dengan menggunakan bahasa pemrograman C++. Sistem diimplementasikan menggunakan komputer dengan spesifikasi sebagai berikut.

1. Perangkat Keras:
   - Prosesor : Intel(R) Pentium(R) CPU P6100
   - RAM (Memori) : 2.00 GB
   - HDD : 320 GB
   - *Recording Device* : Internal Mic
   - *Recording Format* : *up to Studio Quality*
2. Perangkat Lunak:
   - Sistem Operasi : Microsoft Windows 7 Ultimate
   - Bahasa Pemrograman : C++
   - IDE : Microsoft Visual Studio 2008 Professional Edition
   - *Project Types* (IDE) : Visual C++ Win32 Smart Device Project
     Smart Device CAB Project

Hasil implementasi sistem akan dijalankan pada sebuah perangkat *mobile* dengan spesifikasi sebagai berikut.

- *Mobile Phone* : Motorola Q 9h
- *Display* : TFT, 65K colors
  320 x 240 pixels, 2.4"
- Sistem Operasi : Microsoft Windows Mobile 6.0 Standard
- Prosesor : ARM1136OMAP2420
- RAM (Memori) : 96 MB
- ROM : 256 MB
- *Recording Device* : Microphone
- *Recording Format* (ADC) : *Telephone Quality*

4.3.2 Implementasi Perekaman Suara (*Recording*)

Perekaman suara diimplementasikan dengan menggunakan komponen-komponen yang disediakan oleh *Waveform Audio API*. *Waveform Audio API* merupakan salah satu layanan yang disediakan untuk pengembangan aplikasi berbasis *Windows Mobile*. Komponen-komponen pada *Waveform Audio API* dideklarasikan pada *mmsystem.h* (masuk pada *windows.h*) yang merupakan file *header* untuk deklarasi fungsi-fungsi multimedia *Windows*.

Komponen pada *Waveform Audio API* berupa *struct* dan *function* yang digunakan untuk implementasi perekaman suara ditunjukkan pada Tabel 4.4.

Tabel 4.4 Struct dan function pada Waveform Audio API

Struct (Struktur)	
Nama	Deskripsi
WAVEFORMATEX	Mendefinisikan spesifikasi format *recording.*
WAVEHDR	Mendefinisikan spesifikasi serta menyimpan alamat *buffer* yang digunakan untuk menampung data *sample* suara.
Function (Fungsi)	
Nama	Deskripsi
waveInAddBuffer	Mengirimkan atau mengantrikan *buffer* ke perangkat *input* yang digunakan untuk *recording*.
waveInClose	Menutup perangkat *input* yang digunakan untuk *recording*.
waveInOpen	Membuka perangkat *input* yang digunakan untuk *recording*. Fungsi ini juga menentukan mekanisme *callback* yang akan dipanggil setiap kali terjadi *event* (contoh: ketika *buffer* penuh) saat *recording* berlangsung.
waveInPrepareHeader	Mempersiapkan sebuah *buffer* yang dialokasikan pada memori. Fungsi ini berguna untuk memberitahukan perangkat *input* mengenai *buffer* yang akan digunakan untuk menampung data *sample* suara.
waveInReset	Menghentikan proses *recording* dan mengembalikan posisi *record*. Semua antrian *buffer* dikembalikan ke aplikasi dan antrian dimulai dari posisi awal.
waveInStart	Memulai proses *recording*. Selama proses *recording*, setiap kali *buffer* penuh aplikasi akan diberitahu melalui mekanisme *callback*.
waveInStop	Menghentikan proses *recording*. *Buffer* yang sedang diisi dikembalikan ke aplikasi.
waveInUnPrepareHeader	Menghapus persiapan yang telah dilakukan oleh waveInPrepareHeader

(**Sumber :** http://msdn.microsoft.com)

Mekanisme *callback* yang digunakan adalah CALLBACK_WINDOW, sehingga ketika terjadi *event* tertentu, sebuah *message* akan dikirim pada *window*. *Message* tersebut antara lain:

1. MM_WIM_CLOSE

   Message akan dikirim pada *window* ketika perangkat *input* ditutup.

2. MM_WIM_DATA

   Message akan dikirim pada *window* ketika perangkat *input* selesai mengisi *buffer* dengan data *sample* atau ketika *buffer* dikembalikan ke aplikasi.

3. MM_WIM_OPEN

   Message akan dikirim pada *window* ketika perangkat *input* dibuka.

Fungsi *callback* ini dimanfaatkan untuk implementasi pemrosesan data pada aplikasi. Pemrosesan data akan dilakukan pada setiap *buffer* yang telah terisi penuh dan *message* MM_WIM_DATA dikirim pada *window*. *Buffer* juga akan diantrikan kembali untuk proses *recording* sehingga proses *recording* dapat terus berjalan.

Bagian kode C++ untuk perekaman suara adalah sebagai berikut.

1. Deklarasi variabel

```
WAVEFORMATEX waveFormat;
WAVEHDR waveHeader[MAX_BUFFER];
```

2. Mendefinisikan spesifikasi format perekaman suara

```
waveFormat.cbSize = 0;
waveFormat.wFormatTag = WAVE_FORMAT_PCM;
waveFormat.nChannels = nChannels;
waveFormat.nSamplesPerSec = nSamplesPerSec;
waveFormat.wBitsPerSample = wBitsPerSample;
waveFormat.nBlockAlign = nChannels * wBitsPerSample /8;
waveFormat.nAvgBytesPerSec = nSamplesPerSec * nChannels *
wBitsPerSample /8;
```

3. Membuka koneksi dengan *microphone*.

```
mmResult = ::waveInOpen(&hWaveIn, WAVE_MAPPER, &waveFormat,
(DWORD)hwnd, (DWORD)this, CALLBACK_WINDOW);
```

4. Mempersiapkan *buffer*

```
for(int i = 0; i < MAX_BUFFER; i++)
{
      waveHeader[i].lpData = new char[BUFFER_SIZE];
      waveHeader[i].dwBufferLength = BUFFER_SIZE;
      waveHeader[i].dwFlags = 0;
      waveHeader[i].dwUser = i;

      mmResult = ::waveInPrepareHeader(hWaveIn, &waveHeader[i],
sizeof(WAVEHDR));
      if(mmResult != MMSYSERR_NOERROR)
            return FALSE;
      mmResult = ::waveInAddBuffer(hWaveIn, &waveHeader[i],
sizeof(WAVEHDR));
      if(mmResult != MMSYSERR_NOERROR)
            return FALSE;
}
```

5. Memulai perekaman suara

```
mmResult = ::waveInStart(hWaveIn);
```

6. *Callback* pada *window*

```
case MM_WIM_DATA:
      pHdr = ((WAVEHDR*)lParam);
      if(pHdr->dwBytesRecorded != 256)
            return 0;

      if(WHDR_DONE == (WHDR_DONE &pHdr->dwFlags))
      {
            id = pHdr->dwUser;
            bytes = pHdr->dwBytesRecorded;
            if(::waveInAddBuffer(hWaveIn, pHdr, sizeof(WAVEHDR))
== MMSYSERR_NOERROR)
            {
                  switch(mode)
                  {
                  case 0:
                        plot.DrawWave(hWnd, g_hInst,
MAKEINTRESOURCE(IDB_WAVE), (unsigned char const*) pHdr->lpData,
(int)bytes, (int)id);
                        break;
```

```
                    case 1:
                        plot.DrawWave(hWnd, g_hInst,
MAKEINTRESOURCE(IDB_WAVE), (unsigned char const*) pHdr->lpData,
(int)bytes, (int)id);
                        break;
                    case 2:
                        plot.DrawSpectrum(hWnd, g_hInst,
MAKEINTRESOURCE(IDB_SPECTRUM), (unsigned char const*) pHdr-
>lpData, (int)bytes, (int)id);
                        break;
                    case 3:
                        plot.DrawSpectrumLog(hWnd, g_hInst,
MAKEINTRESOURCE(IDB_SPECTRUMDB), (unsigned char const*) pHdr-
>lpData, (int)bytes, (int)id);
                        break;
                    default:
                        break;
                }
            }
        }
        break;
```

4.3.3 Implementasi Algoritma FFT

Pada implementasi algoritma FFT digunakan beberapa *library* yang berada dalam lingkup C++ *Standart Library* yaitu:

1. *Complex*

   Mengimplementasikan kelas *complex* untuk menampung bilangan kompleks, serta fungsi-fungsi dan operator-operator untuk perhitungan bilangan kompleks. *Complex* merupakan sebuah objek yang dideskripsikan sebagai pasangan nilai dengan tipe data tertentu, dengan nilai pertama merepresentasikan komponen *real* dan nilai kedua merepresentasikan komponen *imaginer*. Fungsi dan operator perhitungan yang digunakan antara lain:

   - *abs* : menghitung nilai modulus dari bilangan kompleks
   - *operator** : menghitung perkalian antara 2 bilangan kompleks
   - *operator+* : menghitung penjumlahan antara 2 bilangan kompleks
   - *operator-* : menghitung selisih antara 2 bilangan kompleks

2. *Vector*

   Mengimplementasikan kelas *vector* untuk penampungan urutan data, serta fungsi-fungsi untuk pengoperasian data pada *vector*. *Vector* merupakan *dynamic array* yang digunakan untuk menampung elemen-elemen data dengan tipe tertentu secara terurut, yang memungkinkan akses pada tiap-tiap elemen serta keunggulan-keunggulan lain. Fungsi-fungsi yang digunakan antara lain:

   - *clear* : menghapus elemen-elemen vector
   - *push_back* : menambah sebuah elemen di akhir vector
   - *resize* : mendefinisikan ukuran baru untuk vector
   - *size* : mengembalikan ukuran atau jumlah elemen vector

Bagian kode C++ untuk algoritma FFT adalah sebagai berikut.

1. Deklarasi *header*

```
#include <vector>
#include <complex>
```

2. Deklarasi variabel

```
typedef std::complex<double> Complex;

Complex W;
std::vector<int> bitRev;
std::vector<Complex> X;
```

3. *Bit-reversed*

```
bitRev.resize(fftPoints);
for(int i = 0; i < fftPoints; i++)
{
    int k = i;
    bitRev[i] = 0;
    for(int b = 1; b <= logPoints; b++)
    {
        bitRev[i] <<= 1;
        bitRev[i] |= k & 1;
        k >>= 1;
    }
}
```

4. Menyimpan data *sample* sesuai urutan *bit-reversed* dalam bentuk bilangan kompleks

```
X.resize(fftPoints);
for(int i = 0; i < fftPoints; i++)
        X[bitRev[i]] = Complex((double)(dataSample[i] - 128));
```

5. Menghitung 256-Point FFT

```
int step = 1;
for(int stage = 1; stage <= logPoints; stage++)
{
        int increm = step * 2;
        for(int n = 0; n < step; n++)
        {
            //twiddle factor
            double re = cos(2. * PI * n / increm);
            double im = -sin(2. * PI * n / increm);
            W = Complex(re, im);
            for(int i = n; i < fftPoints; i += increm)
            {
                //2-point butterfly
                Complex T = W;
                T *= X[i+step];
                X[i+step] = X[i];
                X[i+step] -= T;
                X[i] += T;
            }
        }
        step *= 2;
}
```

4.3.4 Implementasi Pembaharuan (*Update*) Tampilan pada Layar

Update tampilan pada layar diimplementasikan dengan menggunakan *Graphic Device Interface* (GDI). GDI merupakan salah satu layanan yang disediakan untuk pengembangan aplikasi berbasis *Windows Mobile*, yang digunakan untuk mengontrol tampilan teks dan grafis. GDI pada *Windows Mobile* memiliki komponen yang terbatas dan tidak mencakup semua komponen ataupun fungsi GDI untuk aplikasi *desktop*.

Komponen pada GDI berupa *struct* dan *function* yang digunakan untuk implementasi *update* tampilan pada layar ditunjukkan pada Tabel 4.5.

Tabel 4.5 *Struct* dan *function* pada GDI

Struct (Struktur)	
Nama	Deskripsi
RECT	Mendefinisikan ukuran panjang dan lebar dari sebuah persegi panjang.
POINT	Mendefinisikan koordinat x dan y dari sebuah titik.
Function (Fungsi)	
Nama	Deskripsi
CreatePatternBrush	Menciptakan sebuah objek *brush* dengan menggunakan pola *bitmap* tertentu.
CreatePen	Menciptakan sebuah objek *pen* dengan spesifikasi tebal, warna dan *style* tertentu.
DeleteObject	Menghapus sebuah objek *pen*, *brush*, *bitmap,* atau objek-objek lain.
FillRect	Mengisi sebuah persegi panjang menggunakan *brush* dengan warna tertentu.
GetClientRect	Mengambil koordinat *client area* (area yang dialokasikan untuk menggambar grafis) dari sebuah *window* (jendela).
GetDC	Mengambil tampilan dari d*evice context* tertentu untuk *client area* dari sebuah *window*. *Device context* adalah struktur GDI yang mendefinisikan teks atau gambar yang akan ditampilkan pada layar atau *device* lain.
LoadBitmap	Memuat gambar *bitmap* dari sebuah file *bitmap* tertentu.
Polyline	Menggambar serangkaian garis-garis yang diperoleh dari penggabungan titik-titik dalam *array* tertentu.
ReleaseDC	Keluar dari sebuah *device context*.
SelectObject	Memilih sebuah objek ke dalam *device contect* tertentu.

(**Sumber :** http://msdn.microsoft.com)

Tampilan pada layar terbagi dalam dua bagian yaitu *background* dan grafik. *Background* merupakan gambar *bitmap* yang menampilkan garis-garis sumbu, skala dan keterangan. Sedangkan grafik adalah hasil pemrosesan data berupa gabungan titik-titik menggunakan *polyline*. Ketika tampilan akan diganti, tampilan lama dihapus dengan cara mengisi *client area* menggunakan *background bitmap*.

Bagian kode C++ untuk *update* tampilan pada layar adalah sebagai berikut.

1. Deklarasi *header*

```
#include <complex>
#include <vector>
```

2. Deklarasi variabel

```
RECT lpRect;
POINT pt;
std::vector<POINT> points;
std::vector<int> sample;
```

3. Fungsi untuk menampilkan grafik sinusoida

```
HDC hdc = ::GetDC(hwnd);
::GetClientRect(hwnd, &lpRect);

HPEN redPen = ::CreatePen(PS_SOLID, 1, RGB(255,0,0));
HBITMAP hBitmap = ::LoadBitmap(hinst,  bmpFile);
HBRUSH hBrush = ::CreatePatternBrush(hBitmap);
::FillRect(hdc, &lpRect, hBrush); //menghapus tampilan lama

sample.resize(dataCnt);
for(int i = 0; i < dataCnt; i++)
      sample[i] = (dataSample[i] - 128); //menyimpan data sample

points.clear();
int yZero = ((lpRect.bottom-lpRect.top) - 1) / 2; //zero point
int xZero = lpRect.left + 45; //zero point
int maxPoints = sample.size();
for(int i = 0; i < maxPoints; i++)
{
      int s = (((sample[i] + 1) * -1) / 2) + yZero;
      pt.x = i + xZero; //koordinat x
      pt.y = s;    //koordinat y
      points.push_back(pt); //menyimpan titik koordinat
}

::SelectObject(hdc, redPen);
::Polyline(hdc, &points[0], points.size()); //menggambar grafik
::DeleteObject(redPen);

::DeleteObject(hBitmap);
::DeleteObject(hBrush);
::ReleaseDC(hwnd, hdc);
```

4. Fungsi untuk menampilkan grafik spektrum frekuensi skala linear

```
HDC hdc = ::GetDC(hwnd);
::GetClientRect(hwnd, &lpRect);

HPEN greenPen = ::CreatePen(PS_SOLID, 1, RGB(0,255,0));
HBITMAP hBitmap = ::LoadBitmap(hinst,  bmpFile);
HBRUSH hBrush = ::CreatePatternBrush(hBitmap);
::FillRect(hdc, &lpRect, hBrush); //menghapus tampilan lama

fft.FfTransform(dataSample); //memanggil fungsi FFT

points.clear();
int yZero = (lpRect.bottom-lpRect.top) - 23; //zero point
int xZero = lpRect.left + 45; //zero point
int maxPoints = dataCnt / 2;
for(int i = 0; i < maxPoints; i++)
{
    int s = (int)(fft.GetMagnitude(i) / 128); //menghitung
nilai magnitude
      pt.x = (i * 2) + xZero; //koordinat x
      pt.y = yZero - s; //koordinat y
      points.push_back(pt); //menyimpan titik koordinat
}

::SelectObject(hdc, greenPen);
::Polyline(hdc, &points[0], points.size()); //menggambar grafik
::DeleteObject(greenPen);

::DeleteObject(hBitmap);
::DeleteObject(hBrush);
::ReleaseDC(hwnd, hdc);
```

5. Fungsi untuk menampilkan grafik spektrum frekuensi skala logaritmik

```
HDC hdc = ::GetDC(hwnd);
::GetClientRect(hwnd, &lpRect);

HPEN bluePen = ::CreatePen(PS_SOLID, 1, RGB(0,0,255));
HBITMAP hBitmap = ::LoadBitmap(hinst,  bmpFile);
HBRUSH hBrush = ::CreatePatternBrush(hBitmap);
::FillRect(hdc, &lpRect, hBrush); //menghapus tampilan lama
```

```
    fft.FfTransform(dataSample); //memanggil fungsi FFT

    points.clear();
    int yZero = (lpRect.bottom-lpRect.top) - 23 - 40; //zero point
    int xZero = lpRect.left + 45; //zero point
    int maxPoints = dataCnt / 2;
    for(int i = 0; i < maxPoints; i++)
    {
        int s = (int)fft.GetMagnitudedB(i); //menghitung nilai
desibel dari magnitude
        pt.x = (i * 2) + xZero; //koordinat x
        pt.y = yZero - s; //koordinat y
        points.push_back(pt); //menyimpan titik koordinat
    }

    ::SelectObject(hdc, bluePen);
    ::Polyline(hdc, &points[0], points.size()); //menggambar grafik
    ::DeleteObject(bluePen);

    ::DeleteObject(hBitmap);
    ::DeleteObject(hBrush);
    ::ReleaseDC(hwnd, hdc);
```

6. Fungsi untuk menghitung nilai *magnitude* dari hasil perhitungan FFT

```
double magnitude = std::abs(X[i]);
```

7. Fungsi untuk menghitung nilai desibel dari *magnitude*

```
if(GetMagnitude(i) == 0)
    return (double)0;
else
    return (double)20*::log10(GetMagnitude(i));
```

4.3.4 Implementasi Menghentikan Aplikasi (*Stop*)

Implementasi menghentikan proses *recording* dilakukan dengan menggunakan fungsi-fungsi *Waveform Audio API*. Bagian kode C++ untuk menghentikan aplikasi adalah sebagai berikut.

1. Menghentikan proses *recording*

```
mmResult = ::waveInStop(hWaveIn);
```

2. Menghapus data pada *buffer*

```
for(int i = 0; i < MAX_BUFFER; i++)
{
        if(waveHeader[i].lpData)
        {
                mmResult = ::waveInUnprepareHeader(hWaveIn,
&waveHeader[i], sizeof(WAVEHDR));
                if(mmResult != MMSYSERR_NOERROR)
                        return FALSE;

                delete[] waveHeader[i].lpData;
                waveHeader[i].lpData = NULL;
        }
}
```

3. *Reset* proses *recording*

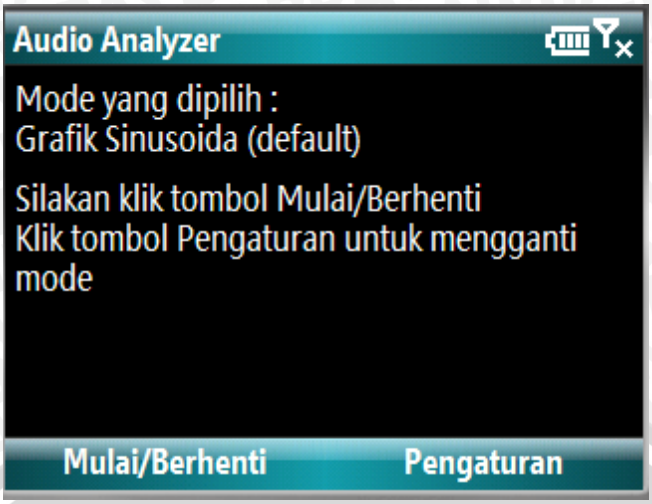
```
mmResult = ::waveInReset(hWaveIn);
```

4. Menutup koneksi dengan *microphone*

```
mResult = ::waveInClose(hWaveIn);
```

4.4 Implementasi Antarmuka (*Interface*)

Pada tampilan aplikasi penganalisis spektrum sinyal digital terdapat dua menu utama yaitu menu mulai/berhenti dan menu pengaturan. Menu mulai/berhenti akan menjalankan fungsi *recording* atau fungsi menghentikan *recording*, tergantung dari kondisi terakhir saat penekanan tombol. Jika tombol ditekan ketika proses *recording* berjalan, maka proses *recording* akan dihentikan dan sebaliknya. Menu pengaturan akan menampilkan sebuah *dialog* yang berisi pilihan-pilihan menu mode tampilan. Tampilan utama aplikasi penganalisis spektrum sinyal digital ditunjukkan pada Gambar 4.17.

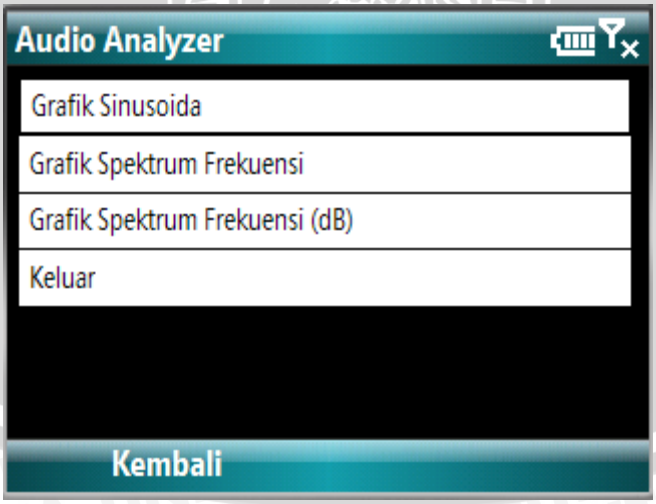


Gambar 4.17 Tampilan utama aplikasi penganalisis spektrum sinyal digital

(**Sumber :** Implementasi)

4.4.1 Implementasi Antarmuka Menu Pengaturan

Menu mode tampilan pada menu pengaturan antara lain yaitu grafik sinusoida, grafik spektrum frekuensi untuk skala linear dan grafik spektrum frekuensi (dB) untuk skala logaritmik. Grafik akan ditampilkan setelah *user* memilih salah satu mode tampilan dan menekan tombol Mulai/Berhenti. Selain menu mode tampilan terdapat menu keluar untuk keluar dari aplikasi (*exit*). Tampilan menu pengaturan ditunjukkan pada Gambar 4.18.

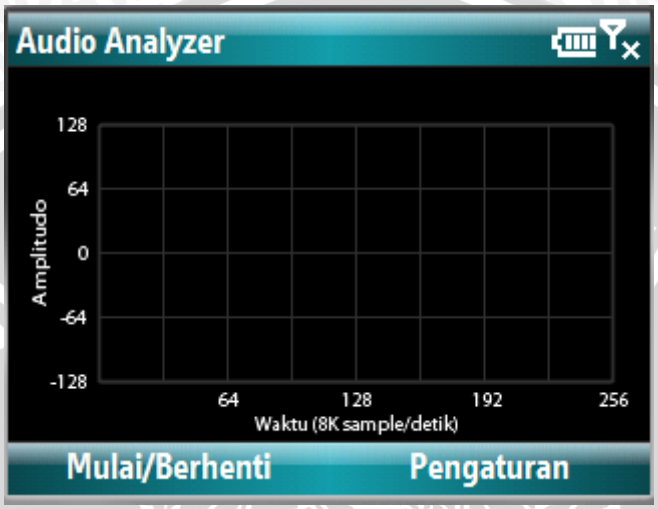


Gambar 4.18 Tampilan menu pengaturan

(**Sumber :** Implementasi)

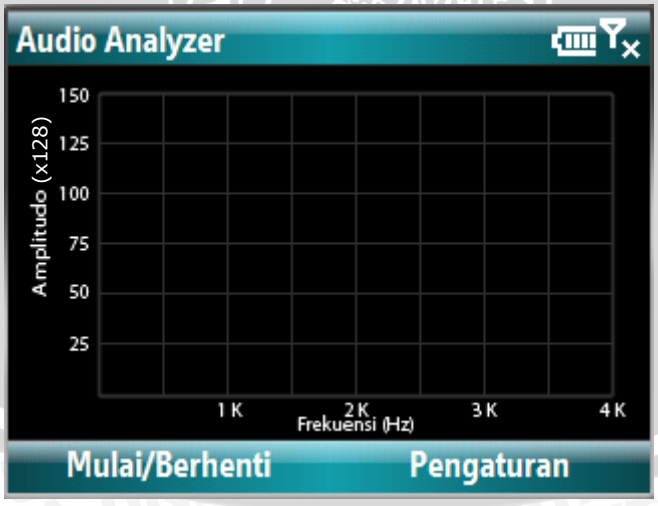
4.4.2 Implementasi Antarmuka Tampilan Grafik

Grafik yang ditampilkan pada aplikasi penganalisis spektrum sinyal digital merupakan hasil dari pemrosesan data yang dilakukan sistem. Tampilan grafik sinusoida, grafik spektrum frekuensi skala linear serta grafik spektrum frekuensi skala logaritmik masing-masing ditunjukkan pada Gambar 4.19, Gambar 4.20 dan Gambar 4.21.



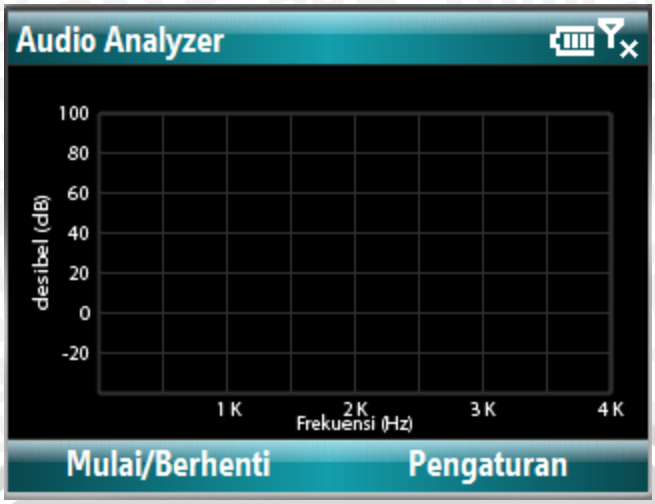
Gambar 4.19 Tampilan grafik sinusoida

(**Sumber :** Implementasi)



Gambar 4.20 Tampilan grafik spektrum frekuensi skala linear

(**Sumber :** Implementasi)



Gambar 4.21 Tampilan grafik spektrum frekuensi skala logaritmik
(**Sumber :** Implementasi)

4.5 Pembuatan Paket Instalasi

Paket instalasi dibangun agar implementasi dari aplikasi dapat dijalankan pada perangkat *mobile phone* dan dapat digunakan oleh *user*. Paket instalasi akan dibuat dalam sebuah *file* CAB (*Cabinet*). CAB adalah ekstensi *file* yang digunakan sebagai format standar *installer* pada aplikasi berbasis *Windows Mobile*.

Paket instalasi CAB *file* dibangun dengan menggunakan salah satu *project* dari Visual Studio 2008 yaitu Smart Device CAB Project. *Project* ini akan ditambahkan pada *solution* tempat *project* utama diimplementasikan. Hasil pembuatan CAB *file* akan ditemukan pada *folder debug* dari Smart Device CAB Project.

Instalasi aplikasi dilakukan dengan menyalin *file* CAB yang telah dibangun ke dalam perangkat *mobile phone* tempat aplikasi akan dijalankan, kemudian melakukan eksekusi terhadap *file* CAB tersebut. Setelah proses instalasi selesai, *shortcut* aplikasi akan muncul pada menu utama perangkat *mobile phone* dan aplikasi dapat dijalankan.

BAB V
PENGUJIAN

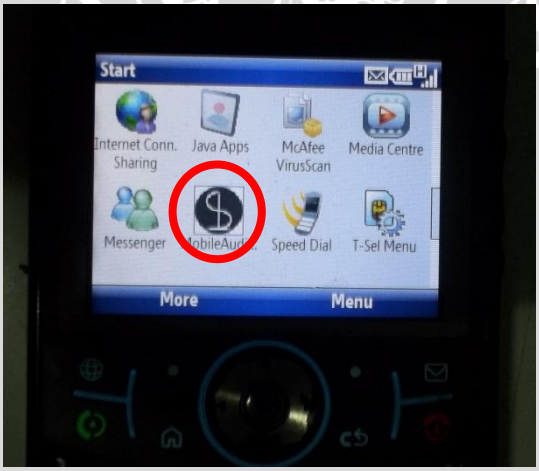
Untuk mengetahui apakah sistem bekerja dengan baik dan sesuai dengan perancangan, maka diperlukan serangkaian pengujian. Pengujian yang dilakukan antara lain:

1. Pengujian hasil tampilan
2. Kompleksitas algoritma
3. Analisis sistem

5.1 Pengujian Hasil Tampilan

Pengujian hasil tampilan dilakukan untuk mengetahui respon aplikasi terhadap sinyal masukan berupa suara. Agar aplikasi dapat dijalankan, instalasi aplikasi dilakukan pada *mobile phone* yang telah ditentukan. Menu *shortcut* yang muncul setelah instalasi ditunjukkan pada Gambar 5.1.

- Besar *file* paket instalasi .CAB : 203.50 KB
- Besar *folder* aplikasi pada *Program Files* : 201.50 KB



Gambar 5.1 Hasil instalasi aplikasi

(**Sumber :** Pengujian)

Pada pengujian hasil tampilan dibutuhkan beberapa komponen. Komponen-komponen yang digunakan untuk pengujian adalah sebagai berikut:

- 1 buah *mobile phone* dimana telah dilakukan instalasi aplikasi.
- 2 buah *speaker* untuk memutar *file* suara.
- 1 buah *file* suara dengan ekstensi .wav yang digunakan sebagai sinyal masukan. *File* ini hanya berisi satu buah nada berdurasi 4 detik yang akan diputar secara kontinyu.
- 1 buah *file* suara dengan ekstensi .mp3.

File suara .wav berupa sebuah nada digunakan sebagai sumber suara *input* agar didapatkan sinyal masukan yang konstan. Pengujian dilakukan tanpa mengetahui nilai komponen frekuensi dan amplitudo dari sinyal masukan. Parameter yang diubah yaitu volume *speaker*. Persiapan pengujian ditunjukkan pada Gambar 5.2.

Gambar 5.2 Persiapan pengujian aplikasi

(**Sumber :** Pengujian)

Pengujian hasil tampilan dilakukan pada beberapa kondisi yaitu:

1. *File* suara (nada) diputar dengan volume 0 (nol)
2. *File* suara (nada) diputar dengan volume rendah (±25% skala)
3. *File* suara (nada) diputar dengan volume maksimum
4. *File* suara (nada) diputar bersamaan dengan *file* suara lain

5.1.1 Hasil Tampilan Grafik Sinyal pada Kondisi 1 (Volume 0/nol)

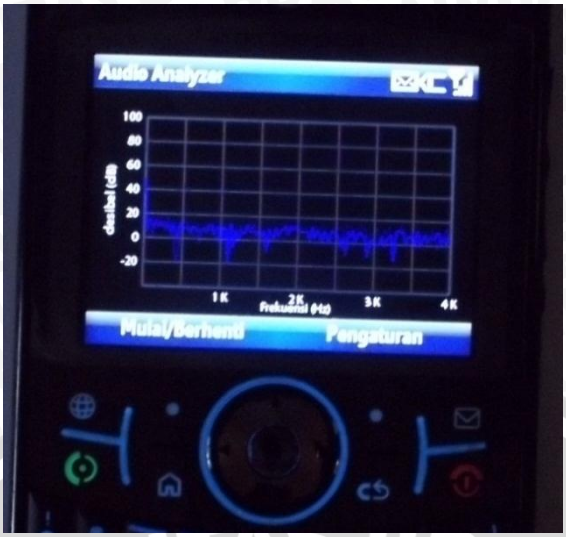
Pengujian hasil tampilan dengan volume 0 (nol) dilakukan pada ruangan yang telah dikondisikan hening, atau tidak ada suara lain yang dapat ditangkap pada jarak jangkauan *mobile phone*. *File* suara diputar dengan volume *speaker* minimum. Hasil tampilan grafik sinusoida, grafik spektrum frekuensi skala linear dan grafik spektrum frekuensi skala logaritmik masing-masing ditunjukkan pada Gambar 5.3, Gambar 5.4 dan Gambar 5.5.

Gambar 5.3 Tampilan grafik sinusoida sinyal pada kondisi 1

(**Sumber :** Pengujian)

Gambar 5.4 Tampilan grafik spektrum frekuensi (linear) sinyal pada kondisi 1

(**Sumber :** Pengujian)



Gambar 5.5 Tampilan grafik spektrum frekuensi (logaritmik) sinyal pada kondisi 1 (**Sumber :** Pengujian)

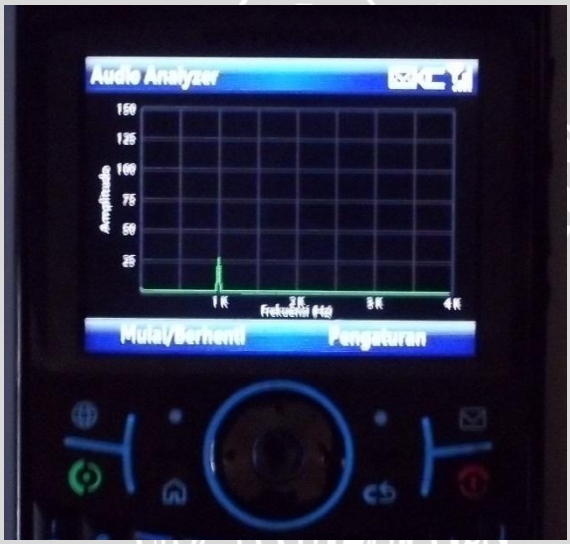
Gambar 5.3 menunjukkan bahwa tidak ada suara yang terekam, sehingga grafik hanya menampilkan nilai amplitudo dengan nilai 0 (nol). Pada Gambar 5.5 terlihat bahwa terdapat nilai-nilai amplitudo yang berada di bawah 20 dB. Nilai-nilai ini merupakan *noise* kuantisasi (*quantization noise*) serta *noise* dari perangkat *recording* yaitu *microphone*, yang dapat dilihat pada *domain* frekuensi. Pada Gambar 5.4 *noise* tidak terlihat karena terjadi perpotongan nilai saat pembagian skala.

5.1.2 Hasil Tampilan Grafik Sinyal pada Kondisi 2 (Volume ±25% skala)

File suara yang digunakan sebagai sinyal masukan diputar dengan volume *speaker* rendahyaitu sebesar ±25% dari skala maksimum. Ruangan tetap dikondisikan agar tidak ada suara lain yang ikut terekam.

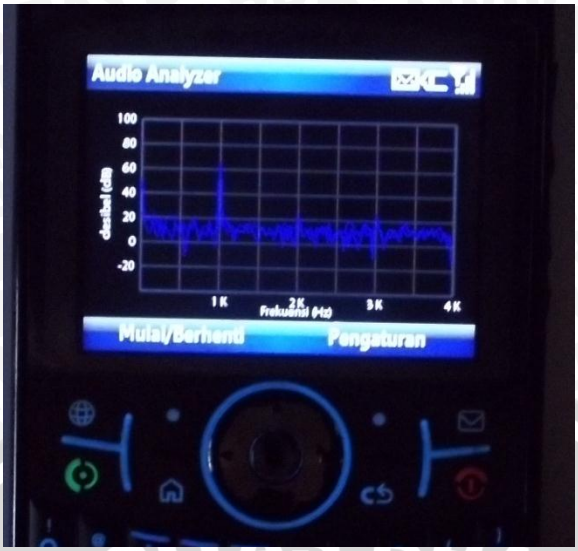
Gambar 5.6 Tampilan grafik sinusoida sinyal pada kondisi 2

(**Sumber :** Pengujian)



Gambar 5.7 Tampilan grafik spektrum frekuensi (linear) sinyal pada kondisi 2

(**Sumber :** Pengujian)



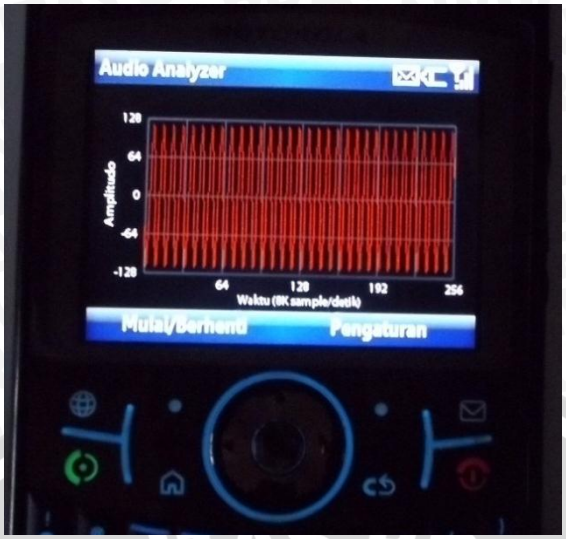
Gambar 5.8 Tampilan grafik spektrum frekuensi (logaritmik) sinyal pada kondisi 2 (**Sumber :** Pengujian)

Hasil tampilan grafik sinusoida, grafik spektrum frekuensi skala linear dan grafik spektrum frekuensi skala logaritmik masing-masing ditunjukkan pada Gambar 5.6, Gambar 5.7 dan Gambar 5.8. Gambar 5.6 menunjukkan bahwa nada yang terekam memiliki bentuk yang konstan dengan nilai amplitudo yang rendah (di antara 0 s/d 64). Komponen frekuensi dari nada ditunjukkan pada Gambar 5.7 dan 5.8 yaitu berada pada frekuensi 1 kHz, dengan nilai amplitudo ±25x128 atau ±60 dB.

Nilai amplitudo pada spektrum frekuensi skala linear maupun spektrum frekuensi skala logaritmik menunjukkan tingkat kenyaringan suara. Semakin tinggi volume, maka suara semakin nyaring. Tingkat kenyaringan suara juga dipengaruhi oleh besarnya jarak sumber suara dari *mobile phone*. Semakin jauh sumber suara, maka tingkat kenyaringan suara semakin rendah dan suara semakin hilang.

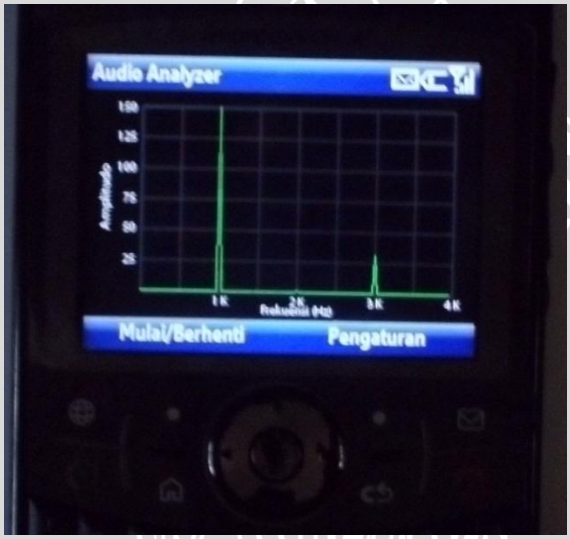
5.1.3 Hasil Tampilan Grafik Sinyal pada Kondisi 3 (Volume maksimum)

File suara yang digunakan sebagai sinyal masukan diputar dengan volume *speaker* maksimum. Ruangan tetap dikondisikan agar tidak ada suara lain yang ikut terekam. Hasil tampilan grafik sinusoida, grafik spektrum frekuensi skala linear dan grafik spektrum frekuensi skala logaritmik masing-masing ditunjukkan pada Gambar 5.9, Gambar 5.10 dan Gambar 5.11.



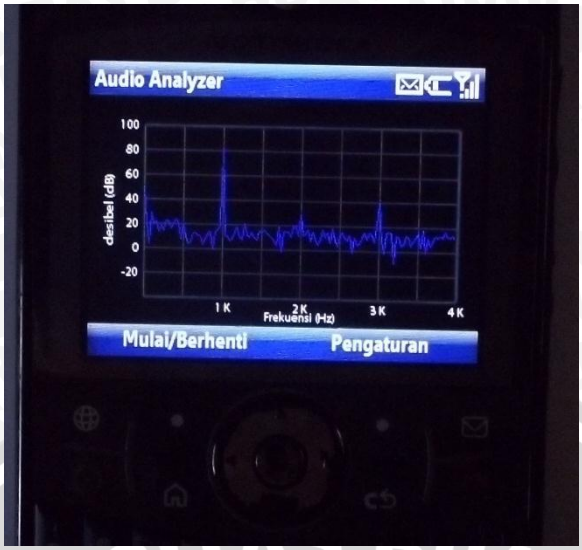
Gambar 5.9 Tampilan grafik sinusoida sinyal pada kondisi 3

(**Sumber :** Pengujian)



Gambar 5.10 Tampilan grafik spektrum frekuensi (linear) sinyal pada kondisi 3

(**Sumber :** Pengujian)

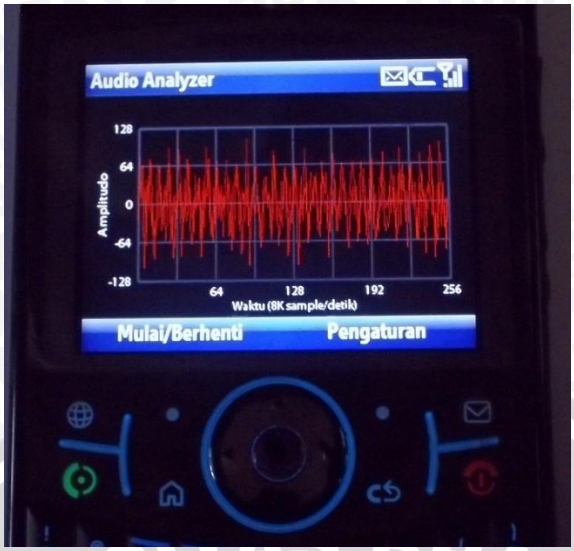


Gambar 5.11 Tampilan grafik spektrum frekuensi (logaritmik) sinyal pada kondisi 3 (**Sumber :** Pengujian)

Gambar 5.9 menunjukkan bahwa nada yang terekam memiliki bentuk yang konstan dengan nilai amplitudo hampir mencapai maksimum (±127). Komponen frekuensi dari nada ditunjukkan pada Gambar 5.10 dan 5.11. Komponen frekuensi tersebut antara lain frekuensi 1 kHz dengan nilai amplitudo ±150x128 (80 dB), serta frekuensi 3 kHz dengan nilai amplitudo ±30x128 (40 dB).

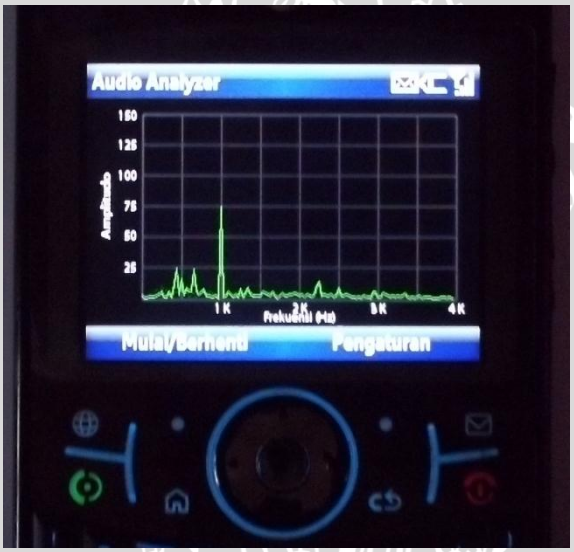
5.1.4 Hasil Tampilan Grafik pada Kondisi 4 (Dua sinyal *input* yang diputar bersamaan)

File suara sinyal masukan dan *file* suara lain diputar secara bersamaan dengan volume *speaker* acak. Volume *file* suara sinyal masukan lebih besar dari *file* suara yang lain. Ruangan tetap dikondisikan agar tidak ada suara lain yang ikut terekam.



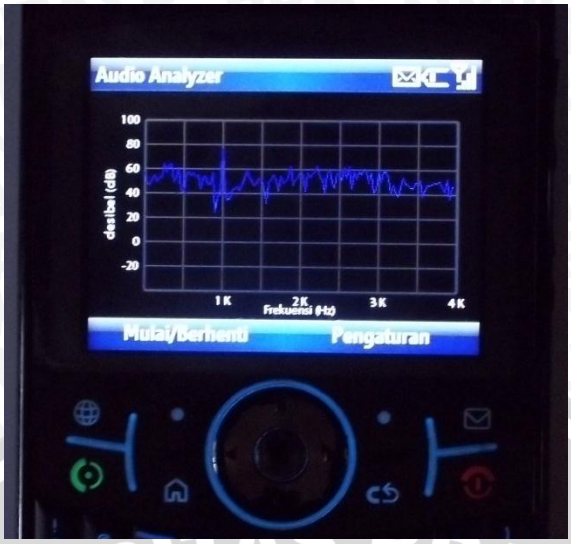
Gambar 5.12 Tampilan grafik sinusoida pada kondisi 4

(**Sumber :** Pengujian)



Gambar 5.13 Tampilan grafik spektrum frekuensi (linear) pada kondisi 4

(**Sumber :** Pengujian)



Gambar 5.14 Tampilan grafik spektrum frekuensi (logaritmik) pada kondisi 4

(**Sumber :** Pengujian)

Hasil tampilan grafik sinusoida, grafik spektrum frekuensi skala linear dan grafik spektrum frekuensi skala logaritmik masing-masing ditunjukkan pada Gambar 5.12, Gambar 5.13 dan Gambar 5.14. Grafik sinusoida tidak memiliki bentuk yang konstan karena merupakan gabungan dari beberapa gelombang suara dengan komponen frekuensi dan amplitudo yang berbeda-beda. Komponen dari gelombang- gelombang suara ini dapat dilihat pada grafik spektrum frekuensi skala linear maupun grafik spektrum frekuensi skala logaritmik.

5.2 Kompleksitas Algoritma

Perhitungan kompleksitas algoritma dilakukan untuk mengetahui besar waktu yang dibutuhkan untuk menjalankan sistem. Kompleksitas algoritma yang dihitung adalah kompleksitas waktu, yang dilakukan dengan cara menghitung banyaknya operasi atau instruksi yang dieksekusi.

Kompleksitas waktu *T(N)* diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan *N*. Ukuran masukan *N* menyatakan jumlah data yang diproses yaitu sebesar 256.

Perhitungan kompleksitas algoritma yang dilakukan yaitu:

1. Kompleksitas algoritma untuk menampilkan grafik sinusoida
2. Kompleksitas algoritma untuk menampikan grafik spektrum frekuensi skala linear
3. Kompleksitas algoritma untuk menampikan grafik spektrum frekuensi skala logaritmik

5.2.1 Kompleksitas Algoritma Grafik Sinusoida

Perhitungan kompleksitas algoritma untuk menampikan grafik sinusoida adalah sebagai berikut.

1. *Komputasi untuk mendapatkan nilai skala pengukuran penuh* → *N*

```
for(int i = 0; i < dataCnt; i++)
    sample[i] = (dataSample[i] - 128);
```

2. *Komputasi untuk mendapatkan nilai koordinat* → *N*

```
for(int i = 0; i < maxPoints; i++)
{
   int s = (((sample[i] + 1) * -1) / 2) + yZero;
   ...
}
```

Jadi total kompleksitas algoritma untuk menampilkan grafik sinusoida adalah *T(N) = 2N*. Dengan mengabaikan nilai konstanta *c = 2*, maka kompleksitas algoritma dapat dinyatakan dalam notasi *big-O* yaitu *O(N)*.

5.2.2 Kompleksitas Algoritma Grafik Spektrum Frekuensi Skala Linear

Perhitungan kompleksitas algoritma untuk menampikan grafik spektrum frekuensi skala linear adalah sebagai berikut.

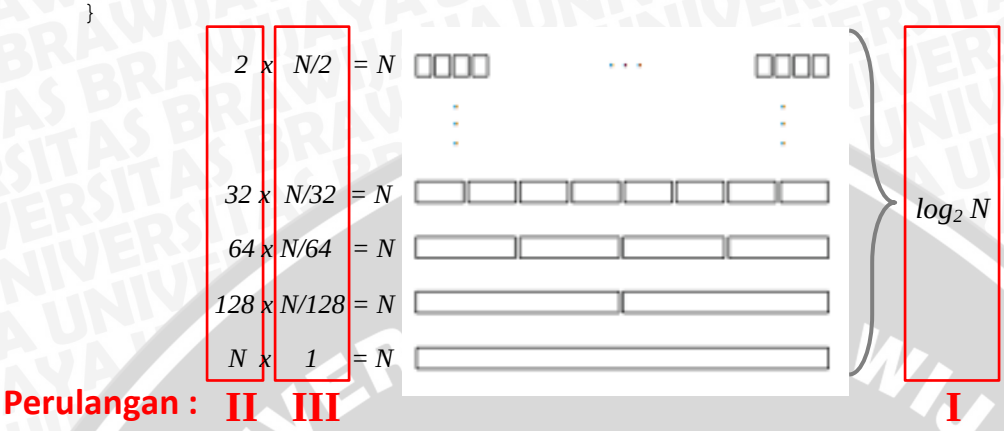
1. *Komputasi untuk menentukan nilai awal perhitungan FFT* → *N*

```
for(int i = 0; i < fftPoints; i++)
   X[bitRev[i]] = Complex((double)(dataSample[i] - 128));
```

2. *Komputasi FFT :*

```
int logPoints = 8;
int step = 1;
for(int stage = 1; stage <= logPoints; stage++)      → Perulangan I
{
   int increm = step * 2;
   for(int n = 0; n < step; n++)                     → Perulangan II
   {     ...
         W = Complex(re, im);
         for(int i = n; i < fftPoints; i += increm)  → Perulangan III
         {     ...
```

```
                X[i+step] -= T;
                X[i] += T;
            }
        }
        step *= 2;
    }
```



Gambar 5.15 Ilustrasi komputasi FFT

(**Sumber :** Pengujian)

$$\text{Komputasi FFT} = \left(\sum_{i=1}^{log_2 N} (2^i . \frac{N}{2^i})\right)$$

$$= \left(\sum_{i=1}^{log_2 N} (N)\right)$$

$$= N(log_2 N) \qquad (5\text{-}1) \qquad \rightarrow Nlog_2 N$$

3. *Komputasi untuk menghitung nilai magnitude* → *N*

```
double magnitude = std::abs(X[i]);
```

4. *Komputasi untuk mendapatkan nilai koordinat* → *N*

```
for(int i = 0; i < maxPoints; i++)
{
   int s = (int)(fft.GetMagnitude(i) / 128);
   ...
}
```

Jadi total kompleksitas algoritma untuk menampilkan grafik spektrum frekuensi skala linear adalah $T(N) = N + Nlog_2N + 2N = Nlog_2N + 3N$. Notasi *big-O* dinyatakan sebagai berikut.

$$Nlog_2 N \rightarrow O(Nlog_2 N) \qquad ; 3N \rightarrow O(N)$$

$$O(max\{Nlog_2 N, N\}) = O(Nlog_2 N) \qquad (5\text{-}2)$$

Kompleksitas algoritma untuk menampilkan grafik spektrum frekuensi skala linear dinyatakan dalam notasi *big-O* yaitu $O(Nlog_2N)$.

5.2.3 Kompleksitas Algoritma Grafik Spektrum Frekuensi Skala Logaritmik

Perhitungan kompleksitas algoritma untuk menampikan grafik spektrum frekuensi skala logaritmik adalah sebagai berikut.

1. *Komputasi untuk menentukan nilai awal perhitungan FFT* → N

```
for(int i = 0; i < fftPoints; i++)
   X[bitRev[i]] = Complex((double)(dataSample[i] - 128));
```

2. *Komputasi FFT* → $Nlog_2N$

```
int logPoints = 8;
int step = 1;
for(int stage = 1; stage <= logPoints; stage++)
{
   int increm = step * 2;
   for(int n = 0; n < step; n++)
   {      ...
          W = Complex(re, im);
          for(int i = n; i < fftPoints; i += increm)
          {      ...
                 X[i+step] -= T;
                 X[i] += T;
          }
   }
   step *= 2;
}
```

3. *Komputasi untuk menghitung nilai magnitude* → N

```
double magnitude = std::abs(X[i]);
```

4. *Komputasi untuk menghitung nilai* desibel *dari magnitude* → N

```
if(GetMagnitude(i) == 0)
   return (double)0;
else
   return (double)20*::log10(GetMagnitude(i));
```

5. *Komputasi untuk mendapatkan nilai koordinat* → N

```
for(int i = 0; i < maxPoints; i++)
{
   int s = (int)(fft.GetMagnitudedB(i) / 128);
   ...
}
```

Jadi total kompleksitas algoritma untuk tampilan grafik spektrum frekuensi skala logaritmik adalah $T(N) = N + Nlog_2N + 3N = Nlog_2N + 4N$. Notasi *big-O* dinyatakan sebagai berikut.

$$Nlog_2N \rightarrow O(Nlog_2N) \quad ; 4N \rightarrow O(N)$$

$$O(max\{Nlog_2N, N\}) = O(Nlog_2N) \tag{5-2}$$

Kompleksitas algoritma untuk menampilkan grafik spektrum frekuensi skala logaritmik dinyatakan dalam notasi *big-O* yaitu $O(Nlog_2N)$.

5.3 Analisis Sistem

Aplikasi dapat bekerja dengan baik dan berjalan sesuai dengan perancangan. *File* paket instalasi cukup kecil karena menggunakan bahasa C++ (*native code*), sehingga tidak membutuhkan banyak memori.

Pengujian hasil tampilan menunjukkan bahwa sinyal masukan dipengaruhi oleh tingkat kenyaringan suara (*sound pressure level*). Volume suara yang rendah dan jarak sumber suara yang jauh menyebabkan tingkat kenyaringan suara yang rendah. Sedangkan volume suara yang tinggi dan jarak sumber suara yang dekat menyebabkan tingkat kenyaringan suara yang tinggi.

Noise yang muncul dapat disebabkan oleh *microphone*, proses pembulatan/kuantisasi (*quantization noise*), dan suara-suara yang tidak diinginkan. Semakin banyak *noise* maka sinyal masukan akan semakin sulit untuk dianalisa.

Kompleksitas algoritma untuk menampilkan grafik sinusoida adalah $O(N)$. Sedangkan kompleksitas algoritma untuk menampikan grafik spektrum frekuensi skala linear maupun skala logaritmik adalah $O(Nlog_2N)$.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, pengujian serta analisis sistem maka dapat diambil kesimpulan sebagai berikut:

1. Pengambilan data dari sinyal masukan berupa suara dilakukan melalui proses perekaman suara (*recording*). Aliran gelombang suara melewati pemrosesan pada *microphone* dan ADC. Data hasil *recording* merupakan data *sample* yang disimpan pada *buffer-buffer* secara bergantian.
2. Perhitungan algoritma FFT dilakukan untuk mengubah data *sample* sinyal dalam *domain* waktu menjadi *domain* frekuensi. Pembentukan algoritma FFT dilakukan melalui penguraian dalam fungsi waktu atau disebut sebagai FFT *Decimation In Time*. Algoritma FFT dapat diilustrasikan menggunakan diagram *butterfly*. Perhitungan algoritma FFT terdiri atas perhitungan *bit-reversed*, perhitungan *twiddle factor*, dan perhitungan *2-point butterfly*.
3. Sistem aplikasi memiliki dua proses yang dijalankan secara hampir bersamaan yaitu ketika *buffer* penuh (*parallel processing*). Proses yang pertama adalah penyimpanan data *sample* (*recording*), sedangkan proses yang kedua adalah pemrosesan data *sample* yang tersimpan pada *buffer*. Proses paralel ini menghasilkan sebuah sistem aplikasi yang bekerja dengan prinsip waktu-nyata (*real-time*).
4. Pengujian hasil tampilan menunjukkan bahwa sinyal masukan dipengaruhi oleh tingkat kenyaringan suara (*sound pressure level*).
5. Nilai level tekanan suara yang dihitung dari nilai *magnitude* hasil perhitungan FFT bukan merupakan nilai akhir dari level tekanan suara. Untuk mendapatkan nilai level tekanan suara yang sebenarnya diperlukan perangkat keras tambahan (*microphone* khusus) dan atau perhitungan lebih lanjut.
6. Kompleksitas algoritma untuk menampilkan grafik sinusoida adalah O(*N)*. Sedangkan kompleksitas algoritma untuk menampikan grafik spektrum frekuensi skala linear maupun skala logaritmik adalah $O(N\log_2 N)$.

6.2 Saran

Dalam perancangan dan pembuatan aplikasi penganalisis spektrum sinyal digital ini masih terdapat kekurangan dan kelemahan, oleh karena itu masih diperlukan adanya penyempurnaan dalam rangka pengembangan ke depan. Adapun hal yang dapat disempurnakan antara lain:

1. Menyempurnakan perhitungan level tekanan suara (SPL) secara lebih akurat dengan memperhitungkan respon frekuensi *microphone*.
2. Menggunakan format *recording* atau *audio quality* yang lebih tinggi (dengan perangkat yang berbeda).
3. Mengimplementasikan analisis spektrum untuk aplikasi pengenal suara (*speech recognition*).
4. Menggunakan sinyal masukan yang berbeda misalnya sinyal *wireless*.

DAFTAR PUSTAKA

_____. *A DFT and FFT Tutorial*. http://www.alwayslearn.com/DFT%20and%20FFT%20Tutorial/DFTandFFT_BasicIdea.html (diakses tanggal 17 September 2011).

_____. *An Introduction to Digital Audio*. http://www.jiscdigitalmedia.ac.uk/audio/advice/an-introduction-to-digital-audio (diakses tanggal 10 September 2011).

_____. *Sound*. http://en.wikipedia.org/wiki/Sound (diakses tanggal 5 Oktober2011).

_____. *Loudspeaker*. http://en.wikipedia.org/wiki/Sound (diakses tanggal 20 Februari 2012).

_____. *Windows Mobile 6.5*. http://msdn.microsoft.com/en-us/library/bb158486.aspx (diakses tanggal 6 Agustus 2011).

Hwei P., Hsu. 1995. *Schaum's Outline of Theory and Problems of Signal and System*. New York : McGraw-Hill.

Milewski, Bartoz. *The Fourier Transform*. http://www.relisoft.com/Science/Physics/sound.html (diakses tanggal 22 April 2011).

Tanudjaja, Harlianto. 2007. *Pengolahan Sinyal Digital & Sistem Pemrosesan Sinyal*. Yogyakarta : Andi.

Wahana Komputer. 2009. *Pemrograman Microsoft Visual C++*. Yogyakarta : Andi.

Wakhid, Muh. Adbul. *Sinyal Analog & Sinyal Digital*. http://abdulmail.blogspot.com/2010/03/sinyal-analog-sinyal-digital.html (diakses tanggal 25 Oktober 2011).

Waluyanti, Sri, dkk. 2008. *Teknik Audio Video*. Direktorat Pembinaan SMK.