BAB II
DASAR TEORI

Bab ini berisi dasar teori yang berhubungan dengan perancangan skripsi antara lain mengenai sistem operasi Android, *Video Graphics Array, framebuffer* Linux, pembesaran gambar dengan *bilinear interpolation* dan jaringan nirkabel *Wi-Fi.*

2.1. Sistem Operasi Android

Android adalah sistem operasi berbasis Linux yang dikembangkan untuk perangkat *mobile* seperti ponsel pintar (*smartphone*) dan komputer tablet [2]. Awalnya Android dikembangkan pada tahun 2003 oleh Android Inc., yang kemudian diakuisisi oleh Google pada tahun 2005. Google kemudian menggandeng berbagai perusahaan perangkat keras, perangkat lunak dan telekomunikasi dan membentuk *Open Handset Alliance,* konsorsium yang bersama-sama membangun dan mengembangkan sistem operasi Android.

Pada peluncuran perdana Android tahun 2007, Android bersama *Open Handset Alliance* menyatakan mendukung pengembangan standar terbuka pada perangkat seluler. Google kemudian merilis kode-kode Android di bawah lisensi Apache, sebuah lisensi perangkat lunak dan standar terbuka pada perangkat seluler. Terdapat dua badan khusus distributor Android yaitu *Google Mail Service* (GMS) yang mendapat dukungan penuh dari Google dan yang tidak didukung oleh Google yaitu *Open Handset Distribution* (OHD).

2.1.1. Arsitektur Android

Android merupakan satu paket lengkap perangkat lunak yang dirancang khusus untuk ditanamkan pada perangkat *mobile.* Di dalam paket tersebut terdapat sistem operasi berbasis Linux, kumpulan *middleware* serta aplikasi-aplikasi inti yang diperlukan untuk pengoperasian perangkat *mobile.* Seluruh perangkat lunak Android dibangun dan didesain dalam sebuah arsitektur yang dikelompokkan dalam beberapa lapisan sebagai berikut:

- ***Applications***

  Lapisan ini merupakan lapisan aplikasi yang terpasang di dalam perangkat *mobile.* Yang termasuk dalam lapisan ini yaitu aplikasi inti dalam perangkat *mobile* seperti kalender, kontak, SMS, *e-mail, browser* dan aplikasi-aplikasi buatan pihak ketiga (*third-party applications*). Lapisan ini merupakan lapisan yang paling sering

diakses oleh pengguna lewat *user interface* (UI). Lapisan ini dijalankan dalam *run-time* Android menggunakan kelas-kelas dan *services* yang disediakan oleh *applications framework*.

- ***Applications Framework***

  Pengembang aplikasi memiliki akses penuh untuk menggunakan API *framework* Android yang juga digunakan pada aplikasi inti. Pengembang bebas untuk mengakses perangkat keras, informasi sumber daya, menjalankan *background service*, mengatur alarm, menambahkan informasi pada *status bar* atau *notifications* atau menggunakan API *framework* yang lainnya. Arsitektur aplikasi dirancang untuk menyederhanakan penggunaan kembali komponen. Seluruh aplikasi dapat mempublikasikan kemampuan dan kapabilitasnya masing-masing dan aplikasi lain dapat menggunakan kemampuan tersebut. Mekanisme yang sama memungkinkan pengguna mengganti komponen-komponen yang dikehendaki. Di dalam semua aplikasi terdapat *service* dan sistem yang meliputi:

  - Seperangkat *View* yang dapat digunakan untuk membangun aplikasi meliputi *List, Grid, Text Boxes, Buttons* dan *embeddedable web browser.*
  - *Content Providers* yang memungkinkan aplikasi untuk mengakses data dari aplikasi lain misalnya *Contacts*, atau untuk membagi data yang dimilikinya.
  - *Resource Manager* yang menyediakan akses ke sumber daya *non-code*, misalnya *localized strings, graphics* dan *layout files.*
  - *Notification Manager* yang memungkinkan semua aplikasi untuk menampilkan *custom alerts* pada *status bar.*
  - *Telephony Manager* yang mengatur seluruh panggilan suara dan memungkinkan aplikasi melakukan panggilan suara.
  - *Location Manager* yang menyediakan akses ke informasi lokasi yang diperoleh dari *Global Positioning System* (GPS) atau informasi dari menara penyedia layanan seluler.
  - *Activity Manager* yang mengatur daur hidup dari aplikasi dan menyediakan *common navigation backstack.*

- ***Libraries***

  Android menyertakan seperangkat *library* atau pustaka dalam bahasa C/C++ yang digunakan oleh berbagai komponen yang ada pada sistem Android. Kemampuan ini dapat diakses oleh *programmer* melewati *Android application framework*. Sebagai

contoh Android mendukung pemutaran format audio, video dan gambar. Berikut ini beberapa pustaka inti di dalam Android:

- *System C Library* yang diturunkan dari *standard C system library* (libc) milik *Berkeley Software Distribution* (BSD), dioptimasi untuk perangkat *embedded* berbasis Linux.
- *Media Libraries* yang berisi pustaka-pustaka untuk *playback* dan *recording* dari berbagai format audio dan video populer serta untuk *encoding dan decoding* gambar, meliputi MPEG4, H.264, MP3, AAC, AMR, JPEG and PNG. *Media Libraries* ini berdasarkan pada *PacketVideo's OpenCORE.*
- *Surface Manager* yang mengatur akses pada *display* dan lapisan *composites 2D* dan *3D graphics* dari berbagai aplikasi.
- *LibWebCore* yang berisi *engine web browser* modern yang mendukung *browser* milik Android maupun *embeddable web view*.
- *3D Libraries* yang digunakan untuk implementasi berdasarkan *OpenGL ES 1.0 APIs*. Pustaka ini menggunakan *hardware 3D acceleration* dan *highly optimized 3D software rasterizer*.
- *FreeType* sebagai penerjemah *bitmap* dan *vector font rendering.*
- *SQLite* sebagai *engine* basis data relasional yang kuat dan ringan yang dapat digunakan oleh semua aplikasi.

- **Android *Runtime***

  Berisi *core libraries* dan *Dalvik Virtual Machine* (DVM). *Core libraries* berisi seluruh pustaka inti pada Java. Android menempatkan sebagian besar fungsi-fungsi yang ada pada pustaka dasar dalam bahasa pemrograman Java ke dalam pustakanya. *Dalvik Virtual Machine* adalah *Java Virtual Machine* yang dioptimasi untuk sistem operasi Android pada perangkat *mobile*. *Dalvik Virtual Machine* dibuat agar perangkat yang digunakan dapat menjalankan *multiple virtual machine*. *Virtual machine* ini akan mengeksekusi berkas dengan format *Dalvik executable* (*.dex*) yang telah dioptimasi untuk penggunaan *memory footprint* yang minimal. *Virtual machine* ini akan menjalankan kelas-kelas yang dikompilasi menggunakan kompiler Java yang selanjutnya akan diubah formatnya menjadi format *.dex*. *Dalvik Virtual Machine* menggunakan *kernel* Linux untuk menjalankan fungsi-fungsi seperti *threading* dan manajemen *low-level memory*.

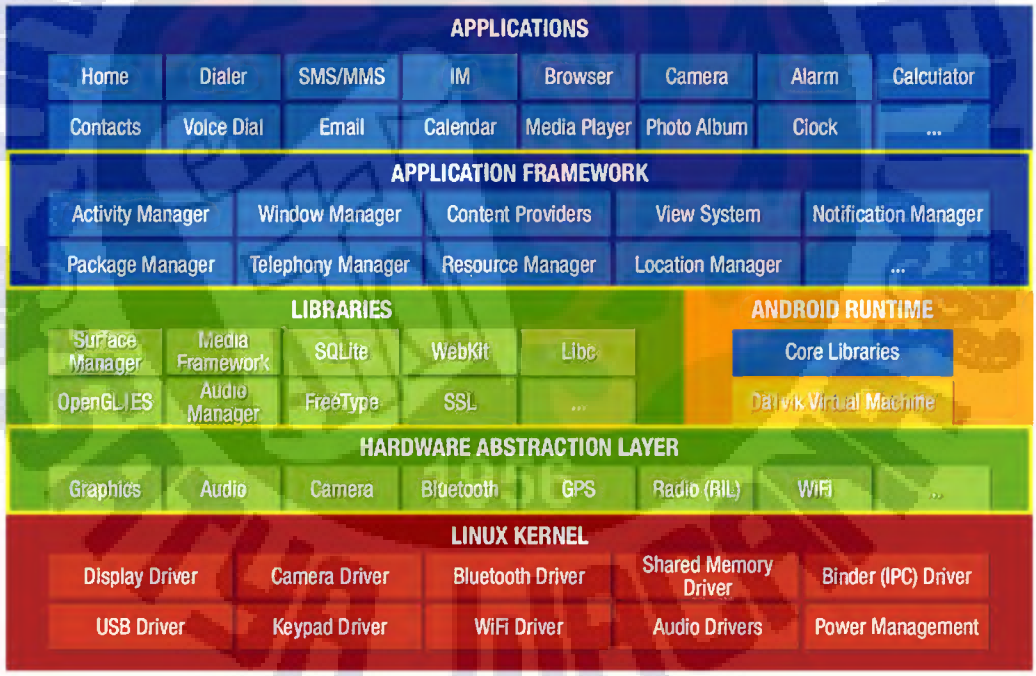
- ***Hardware Abstraction Layer***

  Lapisan ini merupakan penghubung antara *driver* perangkat keras dengan sistem operasi dan sudah ada di dalam *kernel* Linux. Selain itu terdapat juga manajemen sensor seperti GPS, kompas, *accelerometer* dan *proximity*.

- ***Linux Kernel***

  Google menggunakan *kernel* Linux versi 2.6 untuk membangun sistem Android, yang mencakup manajemen proses dan memori, pengaturan keamanan, manajemen catu daya, *network stack* dan *driver* untuk perangkat keras seperti layar, kamera, *keypad, Wi-Fi, Bluetooth, flash memory,* USB, audio dan IPC.

Gambar 2.1 menunjukkan desain susunan lapisan perangkat lunak Android mulai dari lapisan paling rendah sampai lapisan paling tinggi [2].



Gambar 2.1. Arsitektur sistem operasi Android.

2.1.2. Komponen Dasar Aplikasi Android

Setiap aplikasi Android dibangun dari beberapa komponen dasar. Masing-masing komponen tersebut merupakan interaksi antara aplikasi dengan sistem operasi. Tidak semua komponen dasar berinteraksi langsung dengan pengguna, tetapi setiap komponen

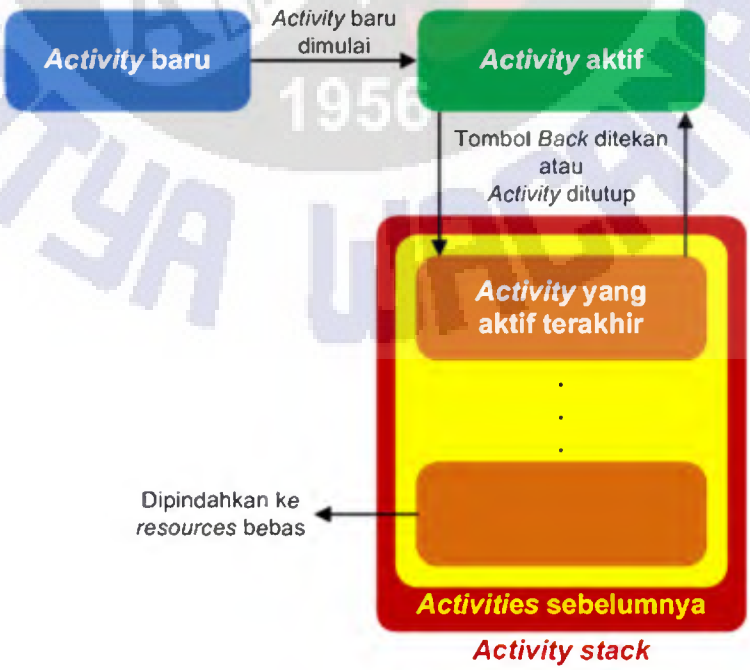
akan berdiri sendiri dan memainkan peranan yang spesifik. Setiap komponen dasar merupakan bagian yang berbeda dan membantu menentukan keseluruhan perilaku aplikasi.

Ada empat jenis komponen dasar aplikasi. Masing-masing komponen memiliki tujuan dan siklus hidup yang berbeda yang mendefinisikan bagaimana komponen tersebut dibuat (*create*) dan dihancurkan (*destroy*).

2.1.2.1. *Activities*

Sebuah *Activity* merupakan komponen yang diwakili oleh sebuah UI visual dalam satu tampilan layar dan digunakan untuk merespon aktivitas tertentu dari pengguna. Sebuah aplikasi dapat memiliki lebih dari satu *Activity* sesuai kebutuhan. Setiap *Activity* memiliki daur hidup masing-masing yang dipengaruhi langsung oleh tugas dan hubungannya dengan *Activity* lain. Selain itu Android memiliki manajemen memori sehingga *Activity* yang tidak aktif dapat dimatikan.

Activity memiliki tiga status dasar yang ditentukan berdasarkan posisinya di *Activity stack* dengan menggunakan mekanisme *last-in-first-out* [1, h.78]. *Activity* baru akan berada di urutan teratas pada *stack* dan ditampilkan di layar menggantikan *Activity* yang sebelumnya ditampilkan di layar. Ketika pengguna menekan tombol *Back* atau *Activity* yang sedang ditampilkan di layar ditutup, *Activity* selanjutnya pada urutan *stack* akan aktif dan ditampilkan di layar. Proses tersebut diilustrasikan pada Gambar 2.2.



Gambar 2.2. Proses di dalam *Activity stack*.

Berikut ini empat status dasar *Activity* saat berada di dalam *Activity stack*:

- ***Active***

  Activity sedang berjalan di *foreground* dan berada di urutan teratas pada *Activity stack. Activity* yang aktif ditampilkan di layar dan dapat menerima masukan dari pengguna. Untuk menjaga agar sebuah *Activity* tetap aktif dan mendapat sumber daya yang dibutuhkan maka sistem operasi Android akan menutup *Activity* yang berada di urutan bawah jika diperlukan.

- ***Paused***

  Saat ada *Activity* lain yang dijalankan di *foreground* dan mendapat fokus dari pengguna. *Activity* yang berstatus *paused* masih ada di memori, masih mempertahankan seluruh informasi di dalamnya dan masih melekat pada *Window Manager,* tetapi tidak ditampilkan di layar atau hanya ditampilkan sebagian dan tidak mendapat fokus. Pada kondisi ekstrim Android akan mematikan *paused Activity* untuk menyediakan sumber daya kepada *Activity* yang sedang aktif, misalnya saat terjadi kekurangan memori pada sistem.

- ***Stopped***

  Saat *Activity* benar-benar tidak terlihat tetapi tetap tersimpan di dalam memori dan mempertahankan seluruh informasi. *Stopped Activity* sudah tidak melekat pada *Window Manager* dan dapat dimatikan dengan mudah saat sistem membutuhkan memori.

- ***Killed***

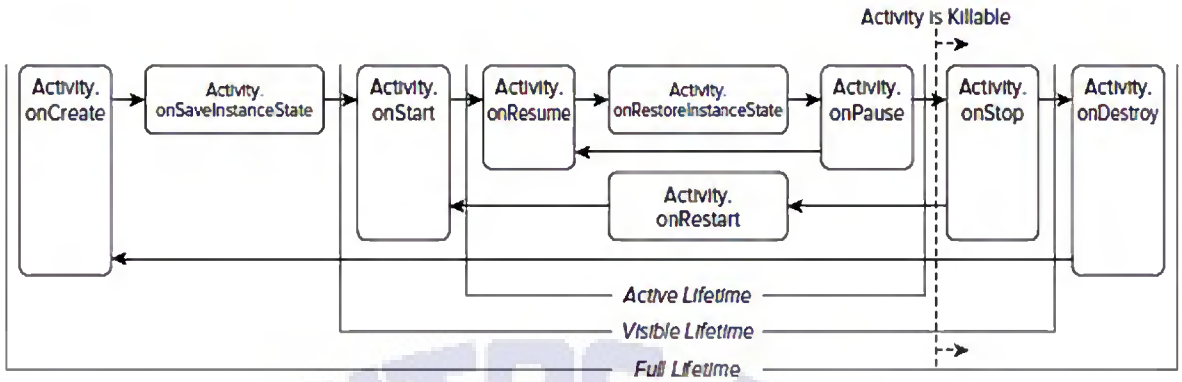
  Jika suatu *Activity* dalam keadaan *paused* atau stopped, sistem dapat mengeluarkan *Activity* dari memori dengan menyelesaikan (*finish*) atau langsung dimatikan (*killed*) prosesnya. Ketika *Activity* tersebut ditampilkan kembali kepada pengguna, maka *Activity* tersebut benar-benar dijalankan dari keadaan awalnya.

Transisi status pada *Activity* adalah *nondeterministic* dan ditangani sepenuhnya oleh manajemen memori Android. Android akan menutup aplikasi yang berisi *Activity* yang tidak aktif dan yang sudah berstatus *stopped*. Pada kasus ekstrim Android juga dapat menutup aplikasi yang berisi *paused Activity*.

Untuk memastikan *Activity* dapat beraksi terhadap perubahan status, Android menyediakan serangkaian *event handler* yang dipanggil saat terjadi transisi *Activity*. *Event handler* ini dibagi berdasarkan masa hidup *Activity* yaitu [1, h.82-84]:

- ***Full lifetime*** terjadi antara pemanggilan awal `onCreate()` dan pemanggilan akhir `onDestroy()`. Dalam beberapa kasus, dimungkinkan proses dari *Activity* dihentikan tanpa pemanggilan `onDestroy()`. `onCreate()` digunakan untuk inisialisasi *Activity* seperti memunculkan UI, mengalokasikan referensi ke variabel-variabel kelas, mengikatkan data ke kontrol dan membuat *Service* dan *Thread*. `onDestroy()` digunakan untuk membersihkan setiap sumber daya yang dibuat dalam `onCreate()`, dan menutup semua koneksi eksternal seperti sambungan ke jaringan atau basis data.
- ***Visible lifetime*** terjadi antara pemanggilan `onStart()` dan `onStop()`. *Activity* yang berada dalam *visible lifetime* akan terlihat oleh pengguna, meskipun tidak mendapatkan fokus atau hanya terlihat sebagian. Dalam beberapa kasus ekstrim, Android akan menghentikan *Activity* selama *visible lifetime* tanpa pemanggilan `onStop()`. `onStop()` digunakan untuk memberikan jeda atau menghentikan proses-proses seperti tampilan animasi, *thread*, *sensor listener*, pencarian GPS, pewaktu, *Service* atau proses lain yang memperbarui UI. Untuk melanjutkan proses-proses tersebut menggunakan `onStart()` atau `onRestart()`. `onRestart()` dipanggil tepat sebelum `onStart()` untuk mengimplementasikan proses tertentu yang harus diselesaikan sebelum proses *Activity* dilanjutkan kembali. `onStart()` atau `onStop()` juga digunakan untuk melakukan *register* dan *unregister* pada *Broadcast Receiver* yang digunakan dalam *Activity*.
- ***Active lifetime*** dimulai dengan memanggil `onResume()` dan diakhiri dengan `onPause()`. *Activity* yang berada dalam *active lifetime* berada di *foreground* dan mendapatkan fokus dari pengguna. Ketika ada *Activity* baru muncul, maka `onPause()` akan dipanggil dan ketika *Activity* kembali ke *foreground* maka `onResume()` akan dipanggil. Pemanggilan `onResume()` dilakukan untuk registrasi ulang pada *Broadcast Receiver* atau proses lain yang ditunda saat `onPause()`.

Gambar 2.3 merangkum seluruh masa hidup *Activity* beserta dengan *event handler*-nya sesuai dengan statusnya di *Activity stack* [1, h.80].



Gambar 2.3. Masa hidup *Activity.*

2.1.2.2. *Services*

Service adalah komponen yang berjalan di latar belakang untuk melaksanakan operasi yang panjang dan tidak memiliki batasan waktu. *Service* tidak menyediakan UI sehingga *Service* akan berjalan tanpa ada interaksi langsung dengan pengguna. *Service* dapat dimulai, dihentikan dan dikendalikan oleh komponen-komponen dasar aplikasi, termasuk *Activities, Broadcast Receivers* atau *Services* yang lainnya. *Service* memiliki dua bentuk umum:

- ***Started***

  Sebuah *Service* dimulai ketika sebuah komponen aplikasi seperti *Activity* memanggil fungsi `startService()`. Setelah dimulai, sebuah *Service* dapat berjalan di latar belakang tanpa ada batasan waktu, bahkan jika komponen yang memulainya dihentikan. Biasanya *Service* yang baru dimulai akan melakukan operasi tunggal dan tidak mengembalikan apapun ke pemanggilnya. *Service* akan menghentikan dirinya sendiri jika tugasnya sudah selesai.

- ***Bound***

  Sebuah *Service* dapat terikat dengan komponen aplikasi dengan memanggil `bindService()`. *Service* yang terikat dengan komponen aplikasi menawarkan antarmuka *client-server* yang memungkinkan komponen untuk berinteraksi dengan *Service*, mengirimkan *request* dan menerima *result* menggunakan *Inter-Process Communication* (IPC). *Service* hanya berjalan selama komponen yang mengikatnya masih berjalan. Beberapa komponen dapat mengikat ke sebuah *Service* sekaligus, dan *Service* akan berhenti ketika seluruh komponen yang mengikatnya dimatikan.

2.1.2.3. *Content Providers*

Content Provider digunakan untuk mengatur dan membagi basis data dengan *Activity* atau *Service* yang lain. Sebuah *Content Provider* menggunakan antarmuka standar dalam bentuk *Uniform Resource Identifier* (URI) untuk memenuhi permintaan data dari aplikasi lain. Android sendiri menyediakan *Content Provider* yang cukup berguna seperti data kalender dan informasi kontak pengguna.

2.1.2.4. *Broadcast Receivers*

Broadcast Receiver merupakan komponen yang menerima dan bereaksi untuk menyiarkan pengumuman. Banyak siaran yang dikeluarkan oleh sistem, misalnya pemberitahuan level baterai rendah, jaringan *Wi-Fi* ditemukan atau pemberitahuan layar mati. Aplikasi juga bisa memulai siaran misalnya memberitahukan aplikasi lainnya bahwa beberapa data sudah selesai diunduh dari Internet dan dapat digunakan.

2.1.3. Android *Software Development Kit*

Android *Software Development Kit* (SDK) menyediakan *Application Programming Interface* (API) *libraries* dan *tools* untuk para pengembang yang diperlukan untuk *build, testing* dan *debugging* aplikasi pada *platform* Android. SDK Android menggunakan bahasa pemrograman Java untuk menulis dan mengkompilasi kode program, oleh karena itu SDK Android harus dilengkapi dengan *Java Development Kit* (JDK).

Karena SDK Android hanya berupa *tools* dan kumpulan API maka dibutuhkan sebuah *Integrated Development Environtment* (IDE) untuk membantu pengembang menulis kode program Android. *Eclipse* merupakan IDE yang umum digunakan untuk menulis kode program Android dan didukung penuh oleh Google sebagai pemiliki lisensi Android. *Eclipse* juga dilengkapi dengan *plugin* khusus yaitu *Android Development Tools* (ADT) yang dirancang untuk mengakomodasi seluruh fitur di dalam SDK Android. ADT memberi kemudahan bagi pengembang dalam pembuatan *project* Android seperti *wizard* untuk membuat *project* baru, membuat *user interface*, menambahkan *package* yang ada di dalam *Android Framework* API dan melakukan *debugging* kode program Android. Untuk keperluan *debugging*, SDK Android menyediakan *Android Virtual Device* (AVD) yang berfungsi sebagai *emulator* untuk sistem operasi Android.

Di dalam SDK Android terdapat API *libraries* yang terdiri dari beberapa level yang dapat dipilih sesuai dengan kebutuhan pengembang. Level API ini akan mempengaruhi

fasilitas dan fitur yang dapat digunakan oleh pengembang dan akan mempengaruhi kompatibilitas dari aplikasi yang dibuat. Tabel 2.1 menunjukkan beberapa versi dan level API tersedia di dalam SDK Android.

Tabel 2.1. Versi *platform* dan level API SDK Android.

Versi *Platform*	Level API	Kode Versi
Android 4.1, 4.1.1	16	JELLY_BEAN
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.3, 2.3.4	10	GINGERBREAD_MR1
Android 2.3, 2.3.1, 2.3.2	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ÉCLAIR_MR1
Android 2.0.1	6	ÉCLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

2.1.4. Android *Native Development Kit*

Walaupun SDK Android menggunakan bahasa pemrograman Java sebagai bahasa pemrograman utama, pengembang juga dapat menggunakan bahasa *native* C/C++ sebagai pelengkap dalam pembuatan aplikasi Android. Pemrograman dengan bahasa *native* C/C++ tersebut terdapat dalam paket *Native Development Kit* (NDK) yang merupakan bagian dari SDK Android. NDK Android berisi seperangkat *cross-toolchains* seperti kompiler dan *linker* serta API *libraries* yang menyediakan *system headers* untuk kode C/C++ [3]. Berikut ini beberapa *system headers* yang tersedia di NDK Android:

- libc (pustaka C) *headers*
- libm (pustaka *math*) *headers*

- JNI *interface headers*
- libz (Zlib *compression*) *headers*
- liblog (Android *logging*) *headers*
- OpenGL ES 1.1 dan OpenGL ES 2.0 (3D *graphics libraries*) *headers*
- libjnigraphics *headers*
- Beberapa *headers* untuk kode C++
- OpenSL ES *native audio libraries*
- *Headers* untuk API *native application*

Penggunaan NDK Android cukup membantu dalam beberapa jenis aplikasi misalnya aplikasi dengan pustaka yang ditulis dalam bahasa C/C++, aplikasi dengan *CPU-intensive operations*, pemrosesan sinyal, *physics simulation* dan sebagainya. Walaupun cukup membantu dalam pengembangan aplikasi Android, tidak semua aplikasi Android harus dilengkapi dengan kode *native* atau bahkan dibangun seluruhnya dengan kode *native*. Penggunaan kode *native* tidak selalu memberikan peningkatan performa yang signifikan pada program, tetapi kode *native* akan menambah kompleksitas pada program karena tidak semua fungsi yang dibutuhkan oleh pengembang aplikasi Android tersedia di *Android Framework API*, misalnya akses ke *framebuffer*.

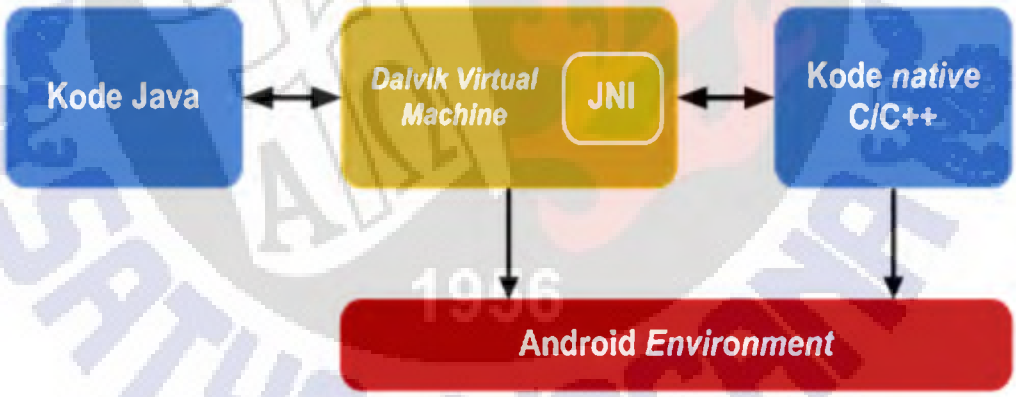
Kode sumber *native* C/C++ dikompilasi dengan menggunakan kompiler dan *linker GNU Compiler Collection* (GCC) yang terdapat pada *cross-toolchains* NDK Android. Hasil kompilasi tersebut dapat berupa berkas *executable, static library* atau *dynamic library* yang dapat dipilih sesuai kebutuhan program. Berikut ini perbedaan antara *executable, static library* dan *dynamic library*:

- ***Executable***: Kode sumber dikompilasi menjadi sebuah *native runnable binary* yang bisa dieksekusi secara mandiri oleh sistem operasi Android.
- ***Static Library***: Kumpulan objek-objek yang terdiri dari variabel, fungsi, kelas atau sumber daya yang dikompilasi menjadi sebuah berkas pustaka. Objek-objek dari *static library* dapat digunakan atau dipanggil dari sebuah kode sumber dan akan ditautkan oleh *linker* pada saat kode sumber dikompilasi. Kode sumber akan dikompilasi bersama dengan *static library* menjadi sebuah *executable* dan objek-objek yang ditautkan akan dimuat pada saat awal program dijalankan.
- ***Dynamic Library***: *Dynamic library* juga berisi objek-objek yang dikompilasi menjadi sebuah berkas pustaka, tetapi *dynamic library* tidak ikut dikompilasi bersama dengan kode sumber melainkan ditautkan dan dimuat secara dinamis pada

saat program berjalan oleh *run-time dynamic linker*. Objek-objek dari *dynamic library* juga dapat digunakan oleh kode sumber lain. Oleh karena itu *dynamic library* juga disebut sebagai *shared library*.

2.1.5. *Java Native Interface*

Java Native Interface (JNI) adalah *framework* pemrograman yang memungkinkan kode Java untuk berinteraksi dengan program atau pustaka yang ditulis dalam bahasa C, C++ dan bahkan *assembly* [4]. JNI merupakan solusi bagi pengembang aplikasi berbasis Java khususnya Android untuk menambahkan fungsi-fungsi yang tidak disediakan oleh API standar pada SDK Android. JNI dapat digunakan pada aplikasi yang memiliki operasi atau kalkulasi yang bersifat *time-critical,* misalnya untuk menghitung persamaan matematika yang rumit dan cukup memakan waktu bila diselesaikan dengan kode Java. JNI juga memungkinkan sebuah aplikasi Android untuk menggunakan *static library* atau *dynamic library* yang ditulis dengan bahasa C atau C++. Gambar 2.4 menunjukkan arsitektur JNI pada aplikasi Android.



Gambar 2.4. Arsitektur JNI.

Karena JNI merupakan antarmuka antara kode Java dengan kode *native* C/C++ dan digunakan untuk saling bertukar data atau informasi, maka JNI menyediakan tipe data yang dapat dimengerti oleh kode *native* C/C++. Tipe data tersebut merupakan konversi dari tipe data primitif dan *reference* pada Java. Tipe data primitif merupakan tipe data yang menyimpan nilai tunggal dan sudah didefinisikan oleh Java. Sedangkan tipe data *reference* merupakan referensi alamat dinamis dari suatu nilai atau objek dan dapat menampung lebih dari satu nilai atau objek (*arrays*). Tipe data *reference* tidak didefinisikan oleh Java

melainkan oleh *user*. Berikut ini tipe data yang disediakan oleh JNI sesuai dengan Tabel 2.2 dan Tabel 2.3.

Tabel 2.2. Tipe data primitif Java dan JNI.

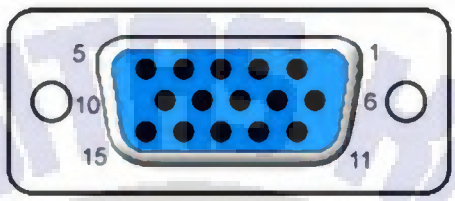
Tipe Data Java	Tipe Data JNI	Lebar Data
`boolean`	`jboolean`	*unsigned* 8 *bits*
`byte`	`jbyte`	*signed* 8 *bits*
`char`	`jchar`	*unsigned* 16 *bits*
`short`	`jshort`	*signed* 16 *bits*
`int`	`jint`	*signed* 32 *bits*
`long`	`jlong`	*signed* 64 *bits*
`float`	`jfloat`	32 *bits*
`double`	`jdouble`	64 *bits*
`void`	`void`	-

Tabel 2.3. Tipe data *reference* JNI dan Java.

Tipe Data JNI	Tipe Data Java
`jobject`	Seluruh *objects* pada Java
`jclass`	Instansiasi dari `java.lang.Class`
`jstring`	Instansiasi dari `java.lang.String`
`jarray`	*Arrays*
`jobjectArray`	`Object[]`
`jbooleanArray`	`boolean[]`
`jbyteArray`	`byte[]`
`jcharArray`	`char[]`
`jshortArray`	`short[]`
`jintArray`	`int[]`
`jlongArray`	`long[]`
`jfloatArray`	`float[]`
`jdoubleArray`	`double[]`
`jthrowable`	*Objects* dari `java.lang.Throwable`

2.2. *Video Graphics Array*

Video Graphics Array (VGA) merupakan standar antarmuka tampilan analog untuk komputer yang dikembangkan oleh IBM dan diperkenalkan pertama kali pada tahun 1987 [5]. Antarmuka VGA menggunakan konektor standar *D-subminiature* atau biasa disebut *D-sub* 15 pin yang dapat dilihat konfigurasinya pada Gambar 2.5.



Gambar 2.5. Konfigurasi pin VGA *female* (keluaran *video card*).

Penjelasan dari masing-masing fungsi pin yang terdapat pada konektor VGA dapat dilihat pada Tabel 2.4.

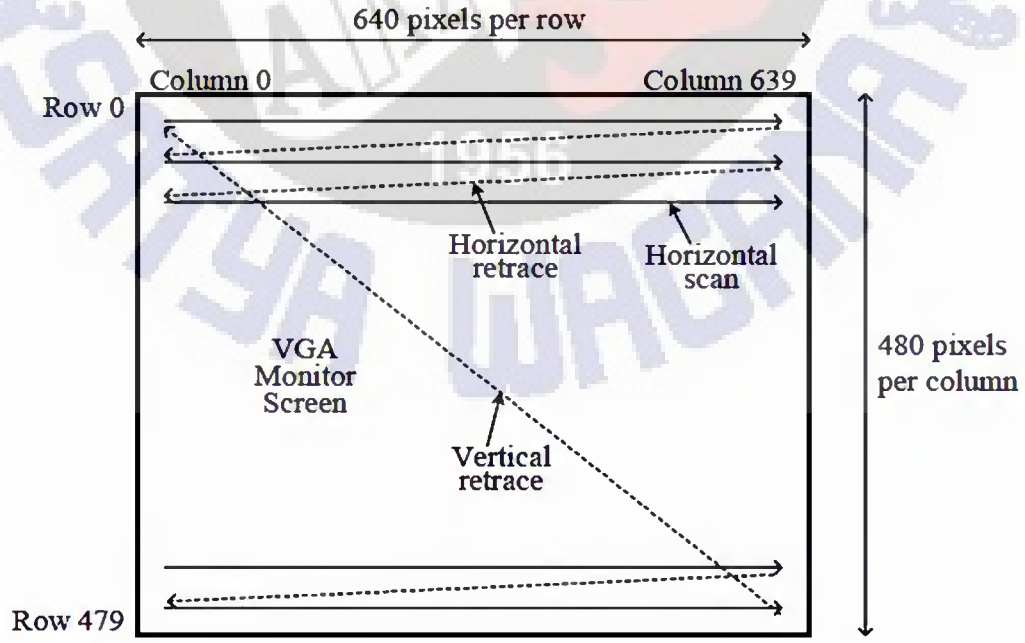
Tabel 2.4. Deskripsi pin VGA.

Pin	Nama Pin	Fungsi
1	RED	*Red Video Output* (75Ω, 0.7Vpp)
2	GREEN	*Green Video Output* (75Ω, 0.7Vpp)
3	BLUE	*Blue Video Output* (75Ω, 0.7Vpp)
4	ID2/Reserved	Monitor-ID Bit 2/*Reserved* (Tidak terkoneksi)
5	GND	*Ground* (0V)
6	RGND	*Red Ground* (0V)
7	GGND	*Green Ground* (0V)
8	BGND	*Blue Ground* (0V)
9	KEY/Power	*Key* (Tidak terkoneksi)/+5V
10	SGND	*Sync Ground*
11	ID0/Reserved	Monitor-ID Bit 0/*Reserved* (Tidak terkoneksi)
12	ID1/SDA	Monitor-ID Bit 1/I^2C *Serial Data*
13	HSync/CSync	*Horizontal Sync/Composite Sync*
14	VSync	*Vertical Sync*
15	ID3/SCL	Monitor-ID Bit 3/I^2C *Serial Clock*

2.2.1. Sinyal Pengontrol VGA

Layar monitor untuk format standar VGA terdiri dari 640 kolom dan 480 baris yang berisi kumpulan piksel. Piksel adalah elemen terkecil yang disusun dalam jumlah tertentu untuk membentuk sebuah gambar, setiap elemen merupakan individu yang berbeda satu sama lain [7, h.1]. Sebuah gambar ditampilkan pada layar dengan kombinasi nyala atau mati pada setiap piksel. Monitor secara kontinu melakukan *scanning* pada keseluruhan layar dengan menyalakan atau mematikan setiap piksel dalam kecepatan tinggi. *Scanning* dengan kecepatan tinggi akan menciptakan kesan semua piksel menyala bersamaan walaupun dinyalakan satu per satu. Jika monitor memiliki kecepatan *scan* yang rendah akan menyebabkan munculnya *flicker* atau kedipan pada layar.

Gambar 2.6 menunjukkan *scanning* untuk layar 640 x 480 piksel yang dimulai dari baris 0, kolom 0 di sudut kiri atas layar dan bergerak ke kanan hingga mencapai kolom terakhir dalam baris tersebut. Ketika *scanning* sudah mencapai kolom terakhir, maka *scanning* meloncat (*horizontal retrace*) ke kolom pertama pada baris berikutnya. Saat *scanning* sudah mencapai piksel terakhir di sudut kanan bawah layar, *scanning* meloncat (*vertical retrace*) dan dimulai lagi dari sudut kiri atas layar. Agar tidak muncul *flicker* pada layar, *scanning* harus dilakukan pada keseluruhan layar minimal 60 kali tiap detik (*refresh rate* 60 Hz). Selama *horizontal* dan *vertical retrace*, seluruh piksel pada layar dimatikan.



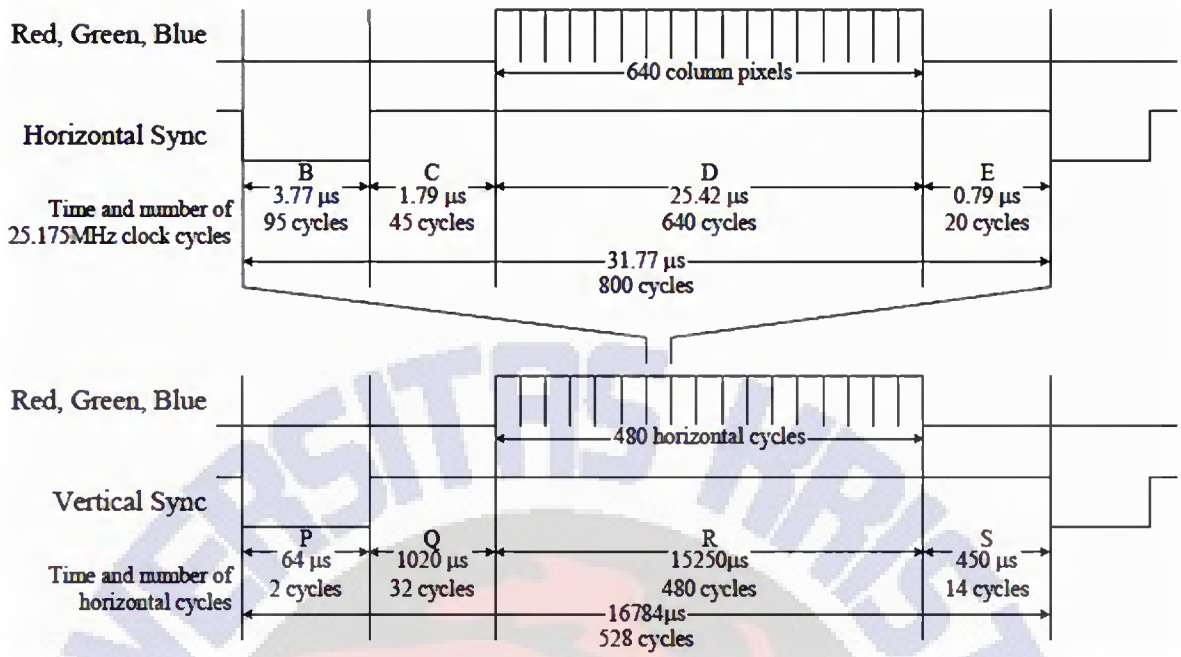
Gambar 2.6. *Scanning* pada layar 640 x 480 piksel.

Antarmuka VGA dikendalikan oleh lima sinyal utama yaitu *red* (merah), *green* (hijau), *blue* (biru), *horizontal synchronization* (*H-sync*) dan *vertical synchronization* (*V-sync*). Sinyal warna atau biasa disebut sinyal RGB mengendalikan warna dari sebuah piksel dalam layar. Sinyal RGB merupakan sinyal analog dengan tegangan maksimum antara 0.7–1 V. Intensitas warna yang berbeda dalam sebuah piksel diperoleh dengan memvariasikan tegangan analog pada sinyal RGB. Variasi tegangan ini dapat diperoleh dengan menggunakan *Digital to Analog Converter* (DAC) berkecepatan tinggi. DAC digunakan untuk merubah nilai warna RGB yang umumnya berupa data digital 5–8 bit untuk tiap warna menjadi tegangan analog.

Sinyal *H-sync* dan *V-sync* digunakan untuk *timing* (pengaturan tempo) kecepatan *scanning*. Kedua sinyal ini merupakan sinyal digital atau sinyal kotak dengan logika '0' (*low*) dan '1' (*high*). Logika *low* memiliki tegangan 0 V dan logika *high* memiliki tegangan +3 V atau +5 V. Panjang sinyal *H-sync* akan menentukan waktu untuk *scanning* baris, sementara *V-sync* menentukan waktu *scanning* keseluruhan layar.

Diagram *timing* dari *H-sync* dan *V-sync* untuk layar 640 x 480 piksel ditunjukkan pada Gambar 2.7. Ketika tidak aktif, baik *H-sync* dan *V-sync* berlogika *high*. *Scanning* baris dimulai dengan membuat *H-sync* menjadi *low* selama 3.77 μs (bagian B) diikuti *high* selama 1.79 μs (bagian C). Selanjutnya nilai RGB dikirimkan untuk setiap piksel dalam suatu waktu, untuk 640 kolom membutuhkan waktu 25.42 μs (bagian D). Setelah nilai RGB piksel ke-640 dikirimkan, sinyal RGB dimatikan dan *H-sync* tetap *high* selama 0.79 μs (bagian E) untuk *horizontal retrace* sebelum *H-sync* menjadi *low* lagi untuk *scan* baris berikutnya. Total waktu untuk *scanning* satu baris penuh yaitu 31.77 μs.

V-sync diberikan logika *low* selama 64 μs untuk mengembalikan *scanning* mulai dari awal lagi (bagian P), kemudian diikuti logika *high* selama 1020 μs (bagian Q). Untuk *scanning* 480 baris dibutuhkan waktu 15250 μs (bagian R) dengan waktu *scanning* tiap baris 31.77 μs seperti yang sudah dijelaskan sebelumnya. Setelah menyelesaikan *scanning* pada baris terakhir, *V-sync* tetap *high* selama 450 μs (bagian S) untuk *vertital retrace* sebelum *V-sync* menjadi *low* lagi untuk memulai *scanning* layar kembali dari awal atau dari sudut kiri atas. Total waktu untuk *scanning* satu layar penuh yaitu 16784 μs.



Gambar 2.7. Diagram *timing H-sync* dan *V-sync* pada layar 640 x 480 piksel [7, h.2].

Pada Gambar 2.7 ditunjukkan untuk *scanning* satu baris penuh dibutuhkan waktu 31.77 µs atau 800 siklus detak (*clock cycle*). Untuk layar 640 x 480 piksel dengan *refresh rate* 60 Hz, nilai frekuensi detaknya adalah 25.175 MHz. Frekuensi detak yang tepat akan menghasilkan *timing* sinyal sinkronisasi yang tepat, serta memudahkan dalam menentukan nilai RGB untuk setiap piksel. Semakin besar resolusi layar, maka frekuensi detak juga semakin tinggi, sebagai contoh layar 800 x 600 piksel dengan *refresh rate* 60 Hz membutuhkan frekuensi detak 40 MHz.

Istilah VGA sendiri pada awalnya mengacu pada tampilan grafis dengan resolusi 640 x 480 piksel, perbandingan panjang dan lebar (*aspect ratio*) 4:3 dan *refresh rate* 60 Hz. Namun saat ini VGA sudah menjadi istilah untuk standar tampilan analog RGB dan digunakan secara luas oleh perangkat komputer maupun perangkat penampil gambar seperti monitor atau proyektor digital. Sejak pertama kali diperkenalkan, antarmuka VGA sudah mengalami banyak perkembangan dan revisi terutama pada resolusi dan *aspect ratio* seperti yang ditunjukkan dalam Tabel 2.5 [6].

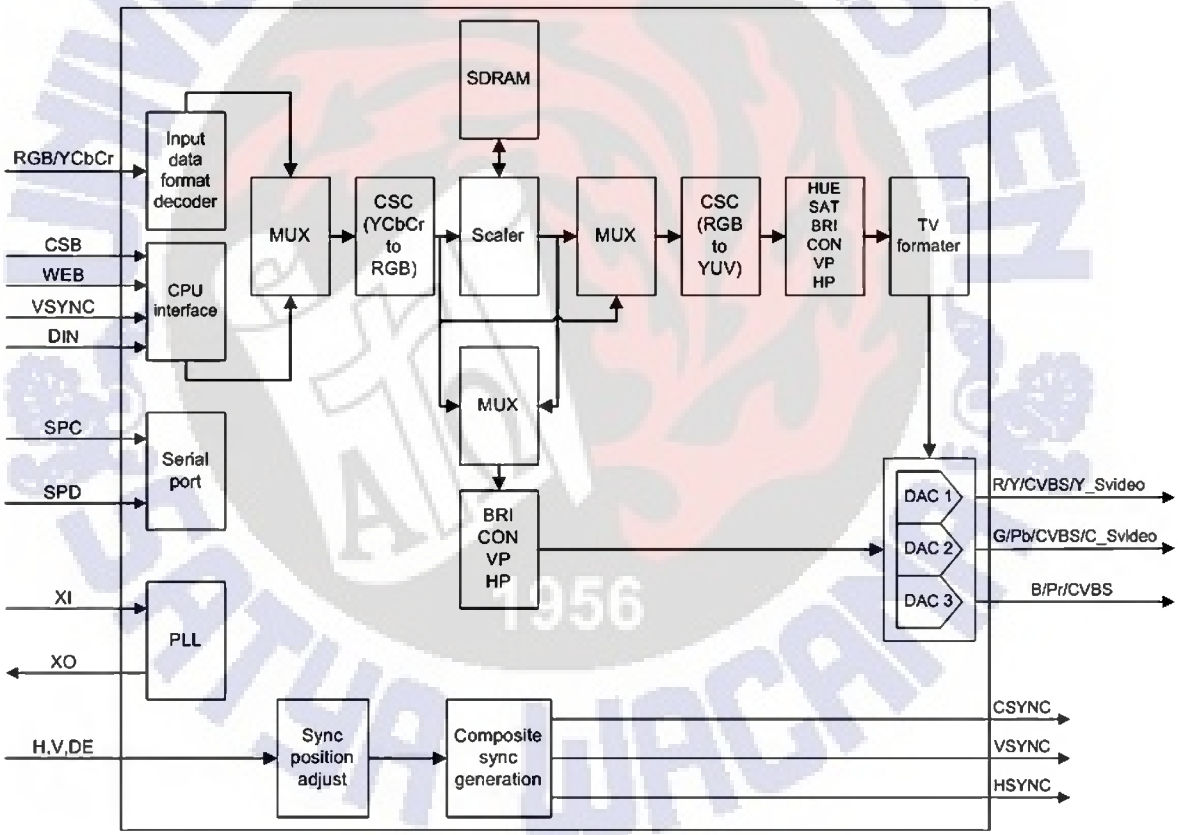
Tabel 2.5. Jenis-jenis resolusi VGA.

Nama	Resolusi (piksel)		*Aspect Ratio*
	Panjang	Lebar	
QQVGA	160	120	4:3
HQVGA	240	160	3:2
QVGA	320	240	4:3
WQVGA	400	240	5:3
HVGA	480	320	3:2
VGA	640	480	4:3
WVGA	800	480	5:3
FWVGA	854	480	16:9
SVGA	800	600	4:3
DVGA	960	640	3:2
WSVGA	1024	576/600	16:9/17:10
XGA	1024	768	4:3
WXGA	1280	768	5:3
XGA+	1152	864	4:3
WXGA+	1440	900	8:5
SXGA	1280	1024	5:4
SXGA+	1400	1050	4:3
WSXGA+	1680	1050	8:5
UXGA	1600	1200	4:3
WUXGA	1920	1200	8:5

Pada perangkat dengan resolusi yang lebih tinggi dari Tabel 2.5 umumnya sudah menggunakan antarmuka digital seperti *Digital Visual Interface* (DVI) atau *High-Definition Multimedia Interface* (HDMI). Penggunaan antarmuka VGA pada resolusi tinggi akan menyebabkan terjadinya penurunan kualitas gambar yang disebabkan oleh keterbatasan jangkauan *timing* pada sinyal analog VGA.

2.2.2. *VGA Encoder* CH7026B-TF

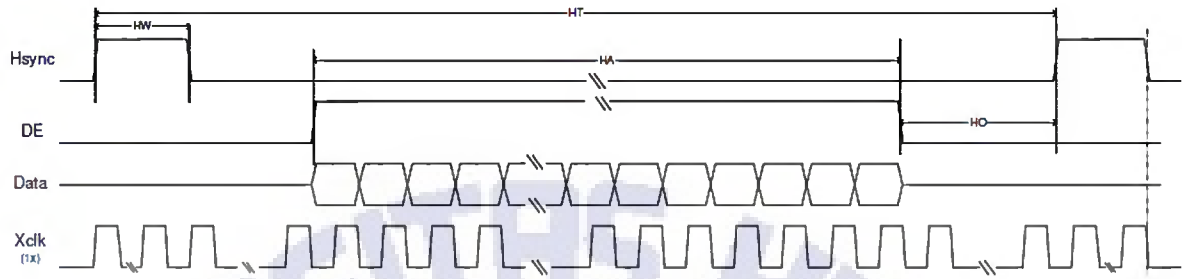
VGA Encoder yang digunakan adalah IC tipe CH7026B-TF keluaran Chrontel yang mampu menghasilkan sinyal pengontrol VGA untuk layar dengan resolusi maksimum 800 x 600 piksel dan kedalaman warna 24-bit (RGB888). CH7026B-TF menerima sinyal masukan digital, menerjemahkannya menjadi format sinyal VGA dan mengirimkan hasilnya melalui tiga kanal DAC yang digunakan untuk keluaran sinyal analog RGB. IC ini juga dapat menghasilkan sinyal sinkronisasi *H-sync* dan *V-sync* sesuai format VGA, serta dilengkapi dengan *Synchronous Dynamic Random Access Memory* (SDRAM) berkapasitas 16 Mbit untuk menyimpan data digital RGB (maksimum 24-bit) yang akan diterjemahkan menjadi sinyal RGB. Gambar 2.8 merupakan blok diagram dari IC CH7026B-TF [17, h.2].



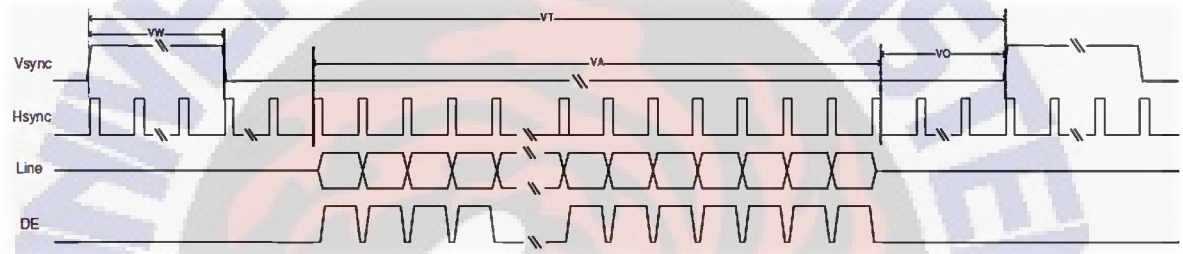
Gambar 2.8. Blok diagram *VGA Encoder* CH7026B-TF.

VGA Encoder CH7026B-TF memiliki 24 pin masukan data RGB, pin *H, V* dan *DE* untuk menghasilkan sinyal dengan format VGA. Pin *H* dan *V* masing-masing berfungsi untuk menerima sinyal sinkronisasi horisontal dan vertikal untuk digunakan bersama dengan masukan data. Pin *DE* berfungsi sebagai indikator masukan data. Jika *DE* diberi *high* maka masukan data aktif, jika *DE* diberi *low* maka masukan data tidak aktif atau

blanking. Gambar 2.9 dan Gambar 2.10 menunjukkan diagram *timing* untuk masukan *VGA Encoder* CH7026B-TF [18, h.12].



Gambar 2.9. Diagram *timing* masukan horisontal CH7026B-TF.



Gambar 2.10. Diagram *timing* masukan vertikal CH7026B-TF.

VGA Encoder CH7026B-TF memiliki 128 *byte* register yang terdiri dari register masukan, register keluaran dan register kontrol, dengan tiap register berukuran 8-bit. Register-register ini dapat diakses melalui antarmuka I^2C dengan kecepatan komunikasi data maksimum 400 kbit/s. Pin yang disediakan oleh IC CH7026B-TF untuk komunikasi I^2C yaitu pin SPC untuk jalur detak dan pin SPD untuk jalur data. Register-register yang ada di dalam IC CH7026B-TF digunakan untuk pengaturan jumlah bit masukan data, konfigurasi format keluaran (SDTV, HDTV atau VGA), resolusi keluaran, *timing* untuk keluaran *H-sync* dan *V-sync* hingga pengaturan *brightness, contrast* dan manajemen catu daya.

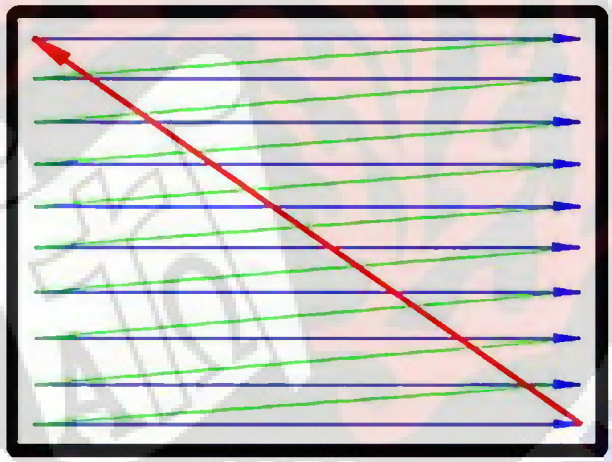
2.3. *Framebuffer* Linux

Framebuffer merupakan lapisan abstraksi pada *kernel* Linux yang menyediakan akses kepada perangkat keras pengontrol grafis [8]. *Framebuffer* memungkinkan perangkat lunak pada sistem operasi Linux menampilkan gambar melalui sebuah antarmuka yang sudah terdefinisi dengan baik. Dengan bantuan *framebuffer* maka perangkat lunak tidak perlu

mengetahui antarmuka *low-level* untuk menampilkan gambar antara *kernel* Linux dengan perangkat keras pengontrol grafis.

Framebuffer dapat diakses dengan pemetaan memori langsung ke memori *framebuffer* pada kartu grafis, atau pemetaan ke ruang memori pada CPU. Ukuran memori yang dipetakan akan menentukan resolusi maksimum dan kedalaman warna pada keluaran grafis. Memori yang sudah dipetakan kemudian diisi dengan nilai warna seluruh piksel dalam gambar yang akan ditampilkan pada layar.

Dengan *framebuffer*, berkas elektron (jika masih menggunakan teknologi layar CRT) digerakkan dari kiri ke kanan dan dari atas ke bawah di seluruh layar. Pada saat yang sama, nilai warna untuk setiap titik pada layar diambil dari memori *framebuffer,* menciptakan satu set elemen gambar diskrit atau disebut piksel. Pola pergerakan berkas elektron pada layar CRT diilustrasikan pada Gambar 2.11.

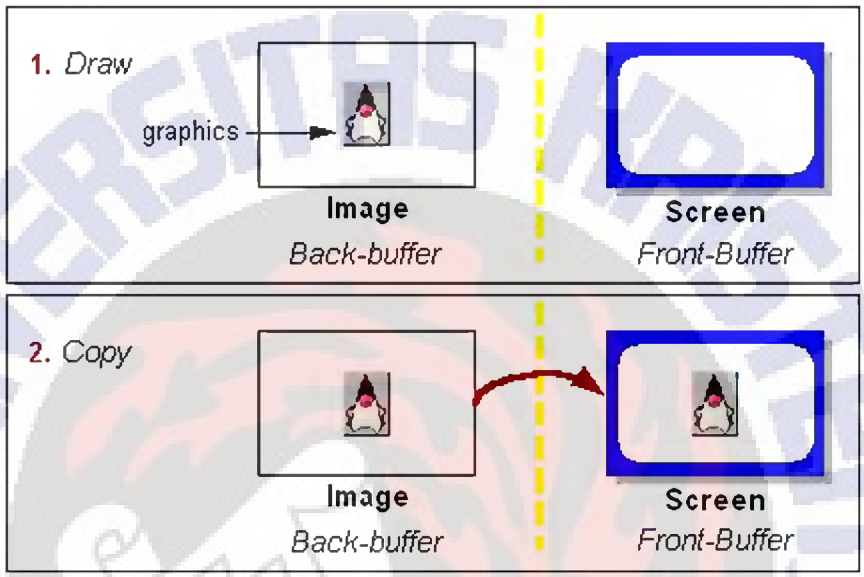


Gambar 2.11. Pola pergerakan berkas elektron pada layar CRT.

Framebuffer sering dirancang untuk menangani lebih dari satu resolusi, oleh karena itu *framebuffer* sering mengandung lebih banyak memori daripada yang diperlukan untuk menampilkan satu *frame* pada resolusi tertentu. Karena memori *framebuffer* memiliki kapasitas yang besar, teknik *double buffering* dikembangkan untuk memungkinkan *frame* baru ditulis ke memori video (*back-buffer*) tanpa mengganggu *frame* yang sedang ditampilkan di layar (*front-buffer*).

Double buffering merupakan teknik untuk *drawing* layar yang mengurangi munculnya *flicker*, sobekan (*tearing*) dan jenis *artifact* lainnya yang menyebabkan tampilan layar menjadi tidak sempurna [9]. Teknik ini juga membuat perpindahan *frame* lebih halus

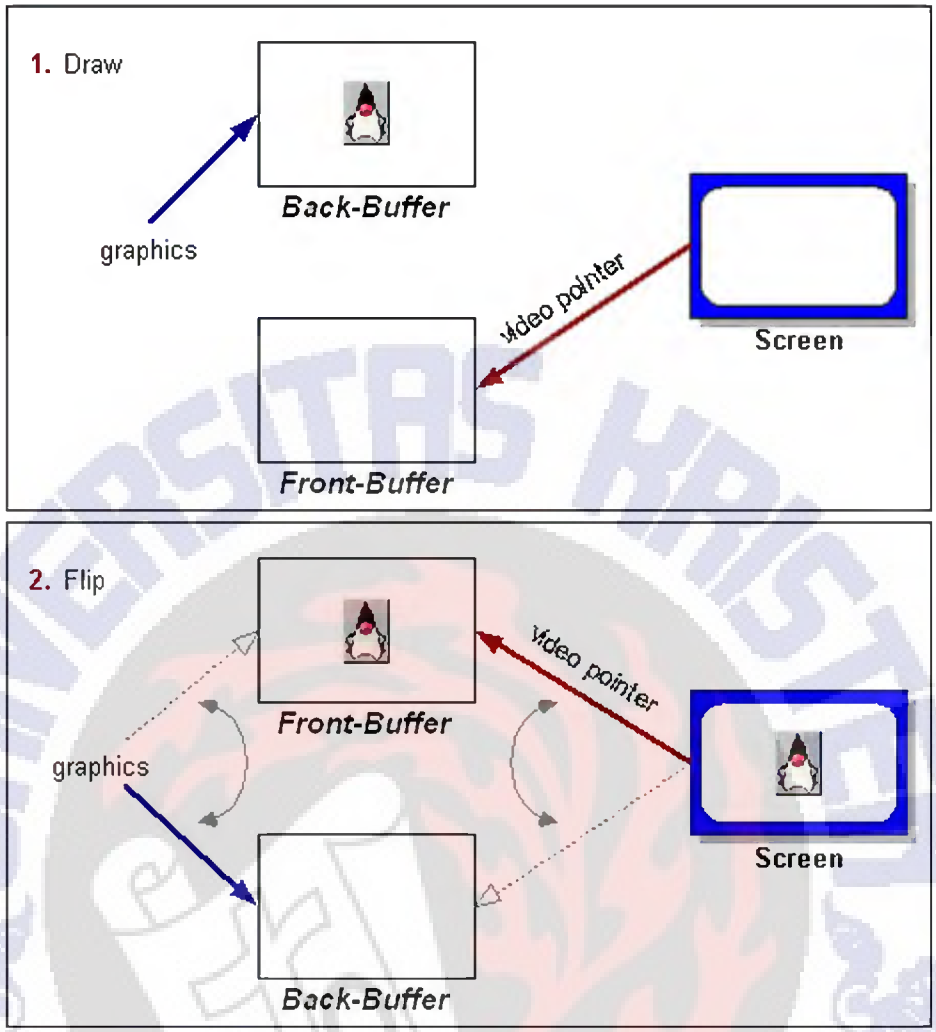
karena pengguna tidak akan melihat proses *drawing* pada layar. Hal ini disebabkan karena *frame* baru ditulis ke *back-buffer* terlebih dahulu sebelum ditampilkan ke layar, baru kemudian disalin ke *front-buffer* dan ditampilkan ke layar, bukan dengan menampilkan *frame* baru langsung pada layar. Konsep penggunaan *double buffering* ditunjukkan pada Gambar 2.12.



Gambar 2.12. Konsep *double buffering* pada layar.

Sistem operasi yang dibangun di atas *kernel* Linux seperti Android dan *Embedded Linux* dilengkapi dengan *framebuffer* yang menggunakan metode *page flipping* untuk transisi antar *frame* yang ditampilkan pada layar. *Page flipping* merupakan variasi dari *double buffering* yang bekerja dengan memindahkan *video pointer* dari *front-buffer* ke *back-buffer,* atau sebaliknya [9]. *Video pointer* akan menunjuk pada blok alamat yang berisi nilai warna setiap piksel yang akan ditampilkan di layar selama *vertical blanking*, atau sesaat sebelum dan sesudah *vertical retrace*.

Frame baru akan ditulis pada *buffer* yang sedang tidak ditunjuk oleh *video pointer*. Saat layar memasuki *vertical blanking*, *video pointer* akan menunjuk ke *buffer* yang berisi *frame* baru, kemudian *buffer* yang sebelumnya ditunjuk oleh *video pointer* digunakan untuk menulis *frame* baru. *Page flipping* lebih cepat dibandingkan dengan *double buffering* biasa yang menyalin seluruh isi *buffer* pertama ke *buffer* kedua. Gambar 2.13 menunjukkan konsep *page flipping* pada layar.

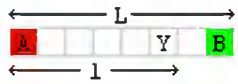


Gambar 2.13. Konsep *page flipping* pada layar.

2.4. *Bilinear Interpolation*

Bilinear interpolation banyak digunakan sebagai algoritma pembesaran gambar karena tidak membutuhkan kalkulasi dan proses yang rumit serta hasil pembesaran yang realistis dan tidak menimbulkan efek pecah pada gambar hasil pembesaran. *Bilinear interpolation* merupakan pengembangan dari *linear interpolation* atau interpolasi untuk memperkirakan nilai dari sembarang titik yang berada di antara dua titik yang diketahui nilainya. *Bilinear interpolation* menggabungkan dua operasi *linear interpolation* untuk memperkirakan nilai dari suatu titik yang berada di antara empat titik terdekat yang diketahui nilainya.

Gambar 2.14 menunjukkan contoh kasus yang dapat diselesaikan dengan *linear interpolation* [11].



Gambar 2.14. *Linear interpolation* untuk mencari warna pada *Y*.

Pada Gambar 2.14 terdapat 8 area sejajar dengan warna merah (RGB = 0xFF0000) di area *A* dan warna hijau (RGB = 0x00FF00) di area *B*. Di antara area *A* dan *B* terdapat 6 area yang masih belum diketahui warnanya, salah satunya yaitu area *Y*. Jarak antara *A* ke *B* yaitu *L* (misal 7) sedangkan jarak antara *A* ke *Y* yaitu l (misal 5). Warna area *Y* dapat dicari dengan persamaan *linear interpolation* sebagai berikut

$$\frac{Y-A}{l} = \frac{B-A}{L}$$

$$Y = A + l\left(\frac{B-A}{L}\right) \quad \text{.....(1)}$$

Karena sebuah warna merupakan campuran dari tiga warna dasar yaitu merah, hijau dan biru, maka untuk menentukan warna dari area *Y* harus dilakukan *linear interpolation* terhadap masing-masing warna dasar sebagai berikut:

- Nilai elemen warna merah pada area *Y* yaitu

$$Y_R = A_R + l\left(\frac{B_R - A_R}{L}\right)$$

$$Y_R = 255 + 5\left(\frac{0-255}{7}\right)$$

$$Y_R \approx 72 = 0x48$$

- Nilai elemen warna hijau pada area *Y* yaitu

$$Y_G = A_G + l\left(\frac{B_G - A_G}{L}\right)$$

$$Y_G = 0 + 5\left(\frac{255-0}{7}\right)$$

$$Y_G \approx 182 = 0xB6$$

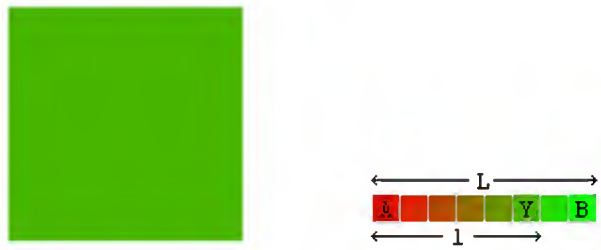
- Nilai elemen warna biru pada area *Y* yaitu

$$Y_B = A_B + l\left(\frac{B_B - A_B}{L}\right)$$

$$Y_B = 0 + 5\left(\frac{0-0}{7}\right)$$

$$Y_B = 0$$

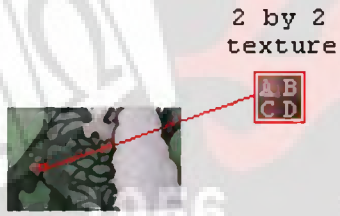
Jadi warna pada area *Y* yaitu RGB = 0x48B600 seperti pada Gambar 2.15 bagian (a) dan warna seluruh area kosong selain area *Y* pada Gambar 2.15 bagian (b).



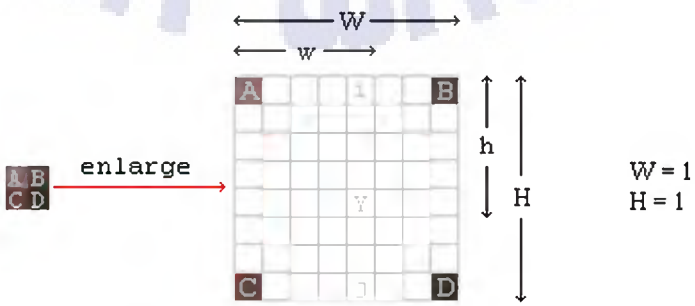
(a) (b)

Gambar 2.15. (a) *Linear interpolation* pada area *Y* dan (b) area lainnya.

Sebuah gambar pada dasarnya tersusun dari banyak tekstur atau bagian-bagian kecil, seperti yang ditunjukkan pada Gambar 2.16. Pembesaran pada sebuah gambar akan menciptakan area atau piksel baru di antara dua piksel yang bersebelahan. Piksel-piksel baru tersebut belum diketahui nilainya, dan jumlahnya ditentukan oleh faktor pembesaran atau perbandingan antara dimensi gambar asli dengan dimensi gambar setelah diperbesar. *Bilinear interpolation* bekerja dengan melakukan *linear interpolation* pada sumbu *x* dan *y* pada tekstur-tekstur 2 x 2 piksel, dan dilakukan pada seluruh tekstur dalam gambar untuk menentukan nilai dari piksel-piksel baru. Gambar 2.17 merupakan contoh pembesaran pada salah satu tekstur dalam sebuah gambar yang diambil dari Gambar 2.16 [11].



Gambar 2.16. Tekstur 2 x 2 yang terdapat dalam sebuah gambar.



Gambar 2.17. Pembesaran pada tekstur 2 x 2 menjadi 8 x 8.

Untuk mencari warna pada area *Y* terlebih dulu dilakukan *linear interpolation* menggunakan Persamaan (1) di atas untuk mencari warna pada area *i* dan *j* yang perhitungannya sebagai berikut:

- Persamaan untuk mencari warna pada area *i* yaitu

$$\frac{i-A}{w} = \frac{B-A}{W}, \text{ dengan } W = 1$$

$$i = A + w\left(\frac{B-A}{W}\right)$$

$$i = A + w(B-A) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2)$$

- Persamaan untuk mencari warna pada area *j* yaitu

$$\frac{j-C}{w} = \frac{D-C}{W}, \text{ dengan } W = 1$$

$$j = C + w(D-C) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

Setelah didapatkan persamaan *linear interpolation* untuk area *i* dan *j* selanjutnya dicari hubungan antara *i*, *j* dan *Y* dengan persamaan sebagai berikut

$$\frac{Y-i}{h} = \frac{j-i}{H}, \text{ dengan } H = 1$$

$$Y = i + h(j-i) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

Persamaan (2) dan (3) kemudian disubstitusi ke dalam Persamaan (4) dan didapatkan sebuah persamaan *bilinear interpolation* sebagai berikut

$$Y = i + h(j-i)$$

$$Y = \big(A + w(B-A)\big) + h\Big(\big(C + w(D-C)\big) - \big(A + w(B-A)\big)\Big)$$

$$Y = A(1-w)(1-h) + B(w)(1-h) + C(h)(1-w) + D(wh) \quad \ldots\ldots\ldots(5)$$

Dengan menggunakan persamaan *bilinear interpolation* sesuai dengan Persamaan (5), dapat ditentukan warna pada seluruh area atau piksel baru yang terbentuk karena pembesaran gambar. Gambar 2.18 menunjukkan hasil pembesaran gambar dengan metode *bilinear interpolation* [11].

Gambar 2.18. Hasil pembesaran gambar dengan *bilinear interpolation.*

2.5. Jaringan Nirkabel *Wi-Fi*

Wi-Fi merupakan kependekan dari *Wireless Fidelity*, yaitu teknologi jaringan nirkabel yang digunakan dalam *Wireless Local Area Networks* (WLAN) sesuai spesifikasi dari *Institute of Electrical and Electronics Engineers* (IEEE) 802.11. Pada dasarnya WLAN sama dengan LAN, hanya saja WLAN tidak menggunakan kabel jaringan seperti LAN untuk menghubungkan setiap komputer yang ada di dalamnya. Jaringan WLAN dengan *Wi-Fi* menggunakan *wireless adapter* yang juga berfungsi sebagai *Network Interface Card* (NIC).

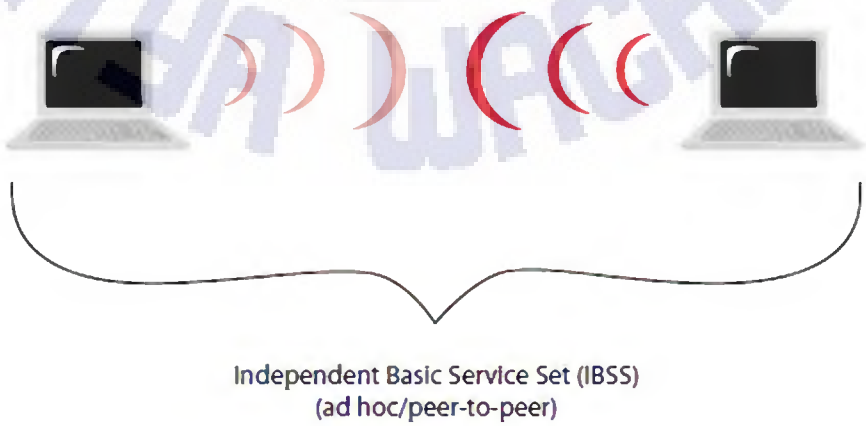
Wi-Fi bekerja dengan memancarkan gelombang radio dengan frekuensi 2.4 GHz atau 5 GHz, tergantung pada standar *Wi-Fi* yang digunakan. Ada empat jenis standar *Wi-Fi* berdasarkan IEEE 802.11 yang dapat dilihat pada Tabel 2.6.

Tabel 2.6. Standar *Wi-Fi* berdasarkan IEEE 802.11.

Standar *Wi-Fi*	Frekuensi	Kecepatan Maksimum	Kompatibilitas
802.11a	5 GHz	54 Mbps	802.11a
802.11b	2.4 GHz	11 Mbps	802.11b
802.11g	2.4 GHz	54 Mbps	802.11b, g
802.11n	2.4/5 GHz	150 Mbps	802.11b, g, n

Spesifikasi IEEE 802.11 mendefinisikan dua jenis mode operasional atau konfigurasi jaringan WLAN yang menggunakan *Wi-Fi*, yaitu:

- **Mode *Ad Hoc* (*Peer-to-Peer*)**

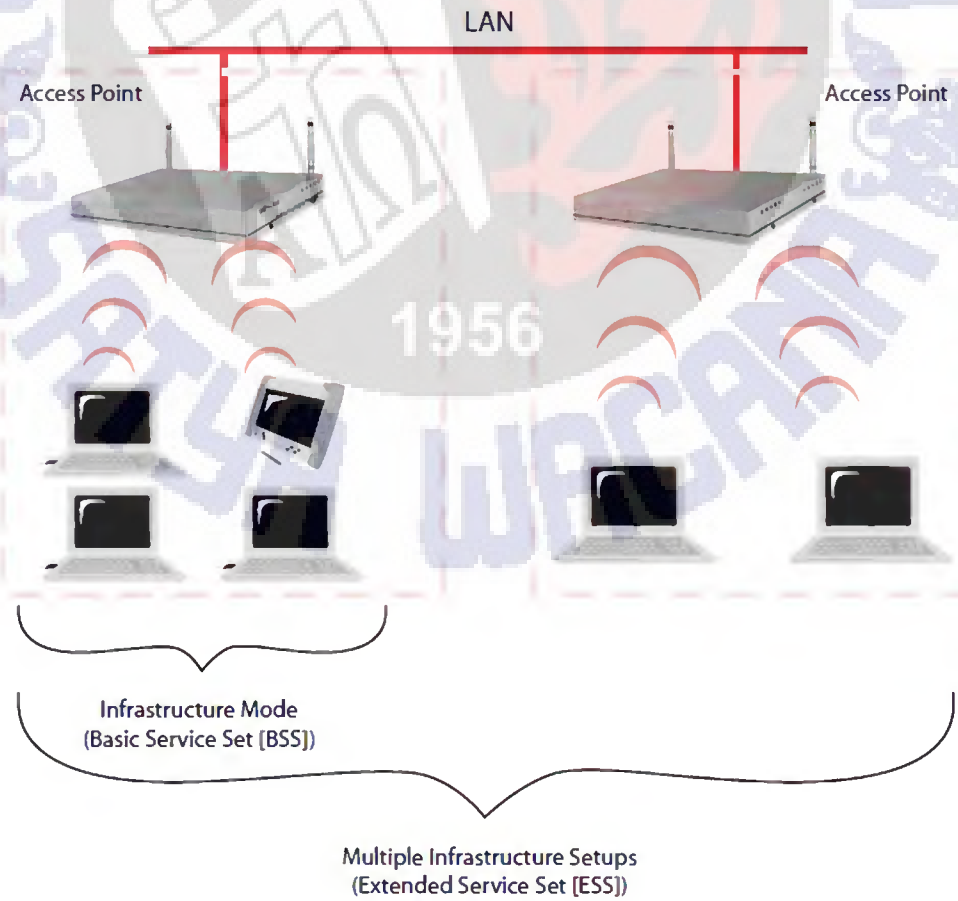


Gambar 2.19. Konfigurasi pada mode *Ad Hoc*.

Gambar 2.19 menunjukkan mode *Ad Hoc* atau juga dikenal sebagai *Independent Basic Service Set* (IBSS) atau mode *peer-to-peer*, jaringan hanya terdiri dari dua *wireless adapter* atau NIC untuk standar 802.11 [14, h.4]. Perangkat yang terhubung dalam jaringan dapat berkomunikasi langsung satu sama lain tanpa menggunakan *wireless access point*.

- **Mode Infrastruktur**

  Pada mode ini jaringan terdiri dari *wireless access point* dan NIC untuk standar 802.11. *Access point* bertindak sebagai *base station* dan semua komunikasi dari anggota jaringan melewati *access point* terlebih dahulu. *Access point* juga dapat digunakan untuk menghubungkan jaringan nirkabel WLAN dengan jaringan LAN. Infrastruktur dasar jaringan nirkabel dengan satu jalur akses disebut *Basic Service Set* (BSS). Bila ada lebih dari satu *access point* yang terhubung ke jaringan untuk membentuk jaringan nirkabel yang lainnya maka infrastruktur ini disebut *Extended Service Set* (ESS). Gambar 2.20 menunjukkan contoh infrastuktur BSS dan ESS [14, h.5].



Gambar 2.20. Infrastruktur BSS dan ESS.