BAB III
PERANCANGAN SISTEM

Dalam skripsi ini akan dirancang perangkat presentasi untuk *smartphone* Android yang dapat ditampilkan pada proyektor digital dengan antarmuka *Video Graphics Array* (VGA). Bab ini akan menjelaskan secara rinci perancangan skripsi yang terdiri dari perancangan perangkat lunak aplikasi *mobile* Android, serta perancangan perangkat *VGA Adapter* baik perancangan perangkat keras maupun perangkat lunaknya.

3.1. Gambaran Sistem

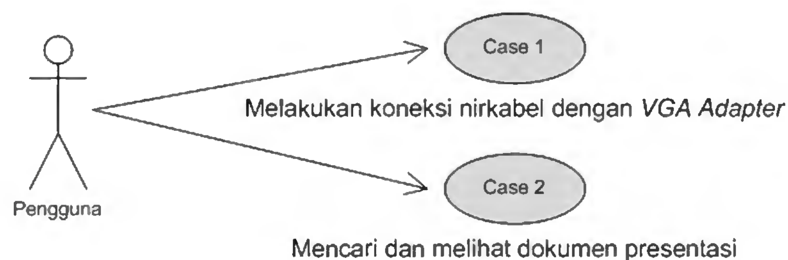
Perangkat presentasi dalam skripsi ini ditujukan untuk *smartphone* Android dan dapat ditampilkan pada proyektor digital dengan antarmuka VGA, sama seperti presentasi dengan komputer atau *notebook*. Sistem presentasi ini merupakan alternatif jika seseorang ingin melakukan presentasi menggunakan *smartphone* Android. Dalam perangkat presentasi ini terdapat aplikasi *mobile* Android untuk menampilkan *slide-slide* presentasi, dan sebuah perangkat penampil gambar yang dapat dihubungkan dengan proyektor digital dengan antarmuka VGA yang selanjutnya disebut *VGA Adapter*.

3.2. Aplikasi *Mobile* Android

Aplikasi *mobile* yang dirancang meliputi *user interface* dengan beberapa *Activity* untuk melakukan koneksi *Wi-Fi* dengan perangkat *VGA Adapter*, menampilkan *slide-slide* presentasi melalui aplikasi *office* pihak ketiga, serta sebuah *Service* untuk menangkap tampilan *slide* presentasi dan dikirim ke perangkat *VGA Adapter*.

3.2.1. *User Interface*

Aktivitas pengguna lewat *user interface* ditunjukkan oleh *use case* pada Gambar 3.1.



Gambar 3.1. *Use case* pengguna.

Penjelasannya adalah sebagai berikut:

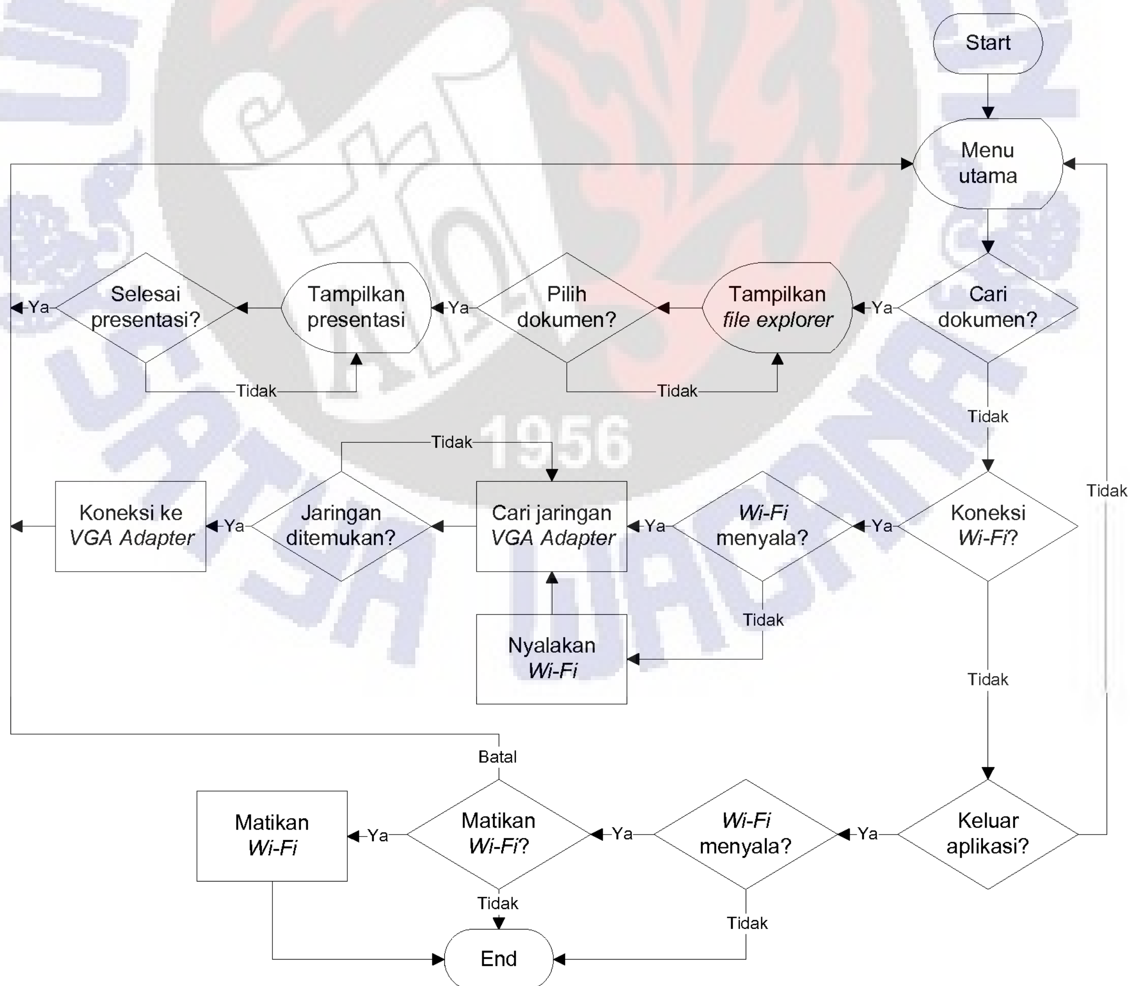
1. Melakukan koneksi *Wi-Fi* dengan *VGA Adapter*

   Pengguna melakukan koneksi dengan *VGA Adapter* melalui jaringan nirkabel *Wi-Fi*. Aplikasi akan mencari jaringan *Wi-Fi* yang disediakan oleh perangkat *VGA Adapter* dan *password* untuk masuk ke jaringan *Wi-Fi* disediakan dan dimasukkan secara otomatis oleh aplikasi.

2. Mencari dan melihat dokumen presentasi

   Aplikasi *mobile* menyediakan sebuah *file explorer* agar pengguna dapat mencari dokumen presentasi yang tersimpan di dalam kartu memori *smartphone* Android. Aplikasi menyediakan akses ke aplikasi *office* pihak ketiga untuk menampilkan dokumen presentasi yang dipilih pengguna lewat *file explorer* tersebut.

Diagram alir dari *user interface* pada aplikasi *mobile* dapat dilihat pada Gambar 3.2.



Gambar 3.2. Diagram alir *user interface* pada aplikasi *mobile* Android.

Penjelasannya adalah sebagai berikut:

1. Ketika aplikasi dijalankan akan muncul menu utama yang berisi pilihan untuk menampilkan *file explorer* atau melakukan koneksi *Wi-Fi* dengan *VGA Adapter*.
2. *File explorer* merupakan tampilan yang berisi daftar *folder* dan konten-konten yang ada di dalam kartu memori *smartphone* Android. Konten yang dapat ditampilkan yaitu dokumen presentasi dengan ekstensi *.ppt, .pptx, .pps, .ppsx*. Jika pengguna memilih sebuah dokumen presentasi maka aplikasi akan memanggil aplikasi *office* pihak ketiga yang terpasang di dalam sistem untuk menampilkan *slide-slide* presentasi.
3. Jika pengguna memilih menu koneksi *Wi-Fi* maka aplikasi akan menyalakan perangkat *Wi-Fi* internal Android terlebih dulu, kemudian aplikasi akan mencari *Service Set Identifier* (SSID) atau nama jaringan *Wi-Fi* dari *VGA Adapter*. SSID *VGA Adapter* memiliki format "`VGA-Adapter_xxxxxx`" dengan "`xxxxxx`" merupakan enam digit terakhir alamat *Media Access Control* (MAC) *Wi-Fi access point* pada *VGA Adapter*. *Password* untuk masuk ke jaringan *Wi-Fi* yaitu "`herditya`" disediakan dan dimasukkan secara otomatis oleh aplikasi *mobile*.
4. Jika pengguna memilih keluar dari aplikasi maka aplikasi akan memeriksa status perangkat *Wi-Fi* internal Android. Jika perangkat *Wi-Fi* masih menyala maka aplikasi akan memberikan pilihan untuk mematikan perangkat *Wi-Fi* sebelum keluar dari aplikasi.

3.2.2. *Screen Capture Service*

Screen capture service merupakan *background worker* yang digunakan untuk menangkap tampilan layar dan disimpan dalam *array* RGB. *Service* ini dimulai saat pengguna membuka dokumen presentasi lewat *file explorer* pada aplikasi *mobile*, bersamaan dengan dimulainya aplikasi *office* pihak ketiga untuk menampilkan *slide-slide* presentasi.

Service ini ditulis dengan bahasa pemrograman *native* C karena SDK Android yang menggunakan bahasa pemrograman Java tidak memberikan akses langsung ke *framebuffer* yang berisi nilai-nilai setiap piksel pada layar. *Framebuffer* Android dapat diakses dengan cara membuka berkas `fb0` pada direktori `/dev/graphics/` pada *filesystem* Android. Fungsi untuk mengakses *framebuffer* ditunjukkan pada Kode 3.1 [10].

```
static int get_framebuffer(GGLSurface *fb) {
     int fd;
     void *bits;

     fd = open("/dev/graphics/fb0", O_RDWR);
     if (fd < 0) {
          perror("Cannot open fb0");
          return -1;
     }

     if (ioctl(fd, FBIOGET_FSCREENINFO, &fi) < 0) {
          perror("Failed to get fb0 info");
          return -1;
     }

     if (ioctl(fd, FBIOGET_VSCREENINFO, &vi) < 0) {
          perror("Failed to get fb0 info");
          return -1;
     }

     bits = mmap(0, fi.smem_len, PROT_READ | PROT_WRITE,
            MAP_SHARED, fd, 0);
     if (bits == MAP_FAILED) {
          perror("Failed to mmap framebuffer");
          return -1;
     }

     fb->version = sizeof(*fb);
     fb->width = vi.xres;
     fb->height = vi.yres;
     fb->stride = fi.line_length / (vi.bits_per_pixel >> 3);
     fb->data = bits;
     fb->format = GGL_PIXEL_FORMAT_RGBA_8888;

     fb++;

     fb->version = sizeof(*fb);
     fb->width = vi.xres;
     fb->height = vi.yres;
     fb->stride = fi.line_length / (vi.bits_per_pixel >> 3);
     fb->data = (void*) (((unsigned) bits) + vi.yres * vi.xres *
               2);
     fb->format = GGL_PIXEL_FORMAT_RGBA_8888;

     return fd;
}
```

Kode 3.1. Fungsi untuk akses *framebuffer*.

Untuk membuka berkas `fb0` digunakan fungsi `open()` dengan argumen pertama adalah lokasi berkas `fb0` dan argumen kedua adalah mode pembacaan. Untuk

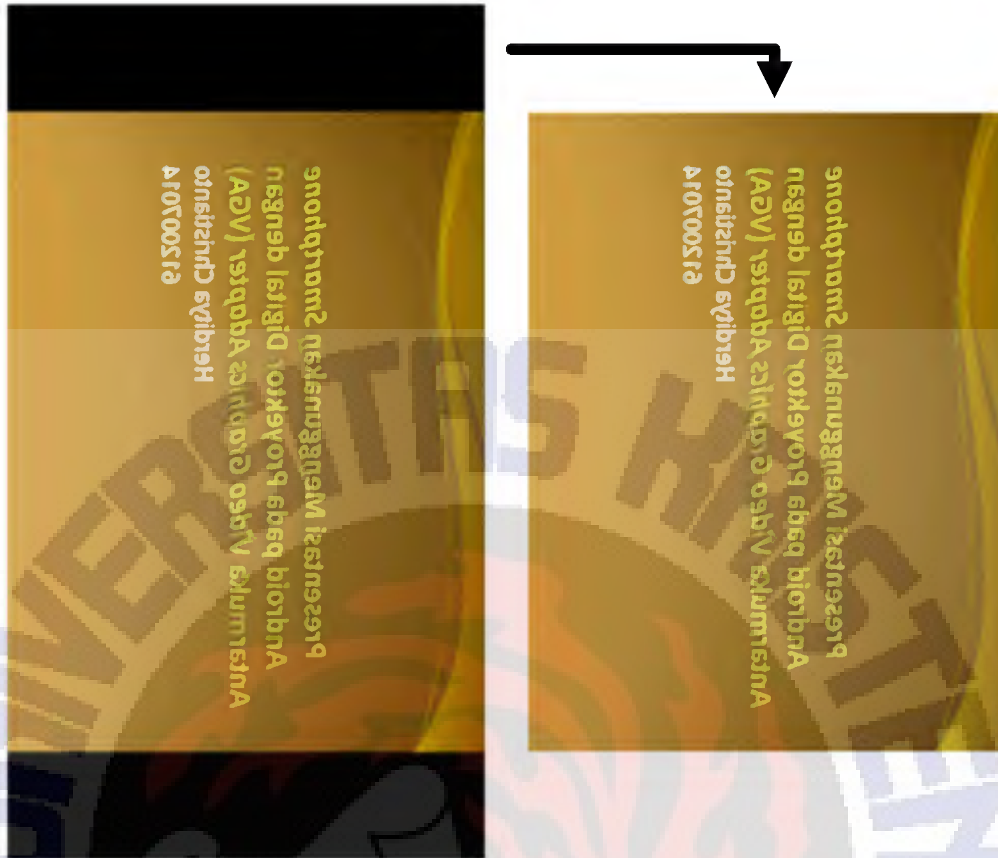
mempermudah proses baca dan tulis pada berkas `fb0` digunakan pemetaan memori program terhadap isi berkas `fb0`, sehingga untuk mengakses *framebuffer* atau isi berkas `fb0` dapat dilakukan seperti mengakses *array* biasa. Pemetaan dapat dilakukan dengan memanggil fungsi `mmap()`. Dalam Kode 3.1 juga terdapat perintah untuk mendapatkan informasi dari layar yang diperlukan untuk proses *screen capture* seperti resolusi, kedalaman warna atau ukuran *framebuffer*.

Setelah proses pembacaan *framebuffer* masih dilakukan beberapa proses transformasi data untuk data dari *framebuffer* yaitu:

1. ***Cropping*** untuk mengambil data yang berisi nilai-nilai warna pada *framebuffer* dan hanya diambil data pada area yang memuat tampilan *slide*. Gambar 3.3 menunjukkan contoh proses *cropping* data *framebuffer* dari layar pada orientasi *portrait* dengan resolusi 480 x 854 piksel (9:16) menjadi 480 x 360 (4:3). Sedangkan Gambar 3.4 menunjukkan contoh proses *cropping* pada orientasi layar *landscape* dengan resolusi 480 x 854 piksel (9:16) menjadi 480 x 640 (3:4).

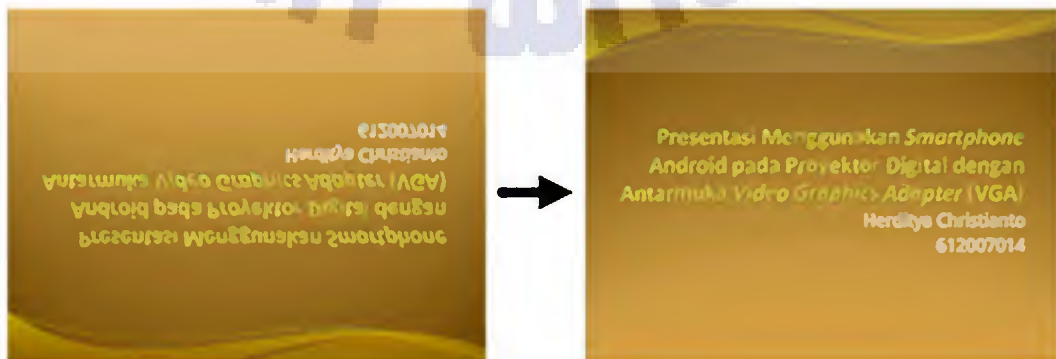
Gambar 3.3. Proses *cropping* pada orientasi layar *portrait*.

UNIVERSITAS KRISTEN SATYA WACANA 1956



Gambar 3.4. Proses *cropping* pada orientasi layar *landscape*.

2. ***Flipping*** untuk membalik data hasil dari *cropping* karena data piksel pada *framebuffer* berada dalam urutan terbalik. Untuk orientasi layar *portrait* dilakukan *flipping* pada sumbu X (*vertical flip*), sedangkan pada orientasi layar *landscape* dilakukan *flipping* pada sumbu Y (*horizontal flip*). Proses *flipping* untuk orientasi layar *portrait* ditunjukkan pada Gambar 3.5 dan untuk orientasi layar *landscape* ditunjukkan pada Gambar 3.6.



Gambar 3.5. Proses *flipping* pada orientasi layar *portrait*.

Gambar 3.6. Proses *flipping* pada orientasi layar *landscape.*

3. ***Rotating*** hanya dilakukan pada data hasil *screen capture* pada orientasi layar *landscape* untuk mendapatkan data gambar dengan *aspect ratio* 4:3. Gambar 3.7 menunjukkan proses *rotating* pada orientasi layar *landscape*.

Gambar 3.7. Proses *rotating* pada orientasi *landscape*.

4. ***Color rearrangements*** merupakan proses akhir sebelum gambar dikirimkan ke perangkat *VGA Adapter* seperti pada Gambar 3.8. Proses ini dilakukan karena *framebuffer* menyimpan data gambar dengan urutan warna ABGR (*Alpha-Blue-Green-Red*). Proses ini merubah urutan warna ABGR menjadi RGB dan menghilangkan kanal warna *Alpha*.

Gambar 3.8. Proses *color rearrangements.*

Seluruh proses *cropping, flipping, rotating* dan *color rearrangements* untuk data *framebuffer* pada orientasi layar portrait ditunjukkan pada Kode 3.2 dan untuk orientasi layar *landscape* ditunjukkan pada Kode 3.3.

```
int w = vi.xres, h = vi.yres, h_aspect, crop;
int x, y, offset = 0;
uint32_t pixel32;

uint8_t *frame = (uint8_t *) malloc(480 * 360 * 3);

h_aspect = (w / 4) * 3;
crop = (h - h_aspect) / 2;
for (y = crop; y < (crop + h_aspect); y++) {
     for (x = 0; x < w; x++) {
          pixel32 = ((uint32_t *) gr_framebuffer[0].data)[(480 *
                         y) + x];
          frame[3 * offset + 0] = pixel32;
          frame[3 * offset + 1] = pixel32 >> 8;
          frame[3 * offset + 2] = pixel32 >> 16;
          offset++;
     }
}
```

Kode 3.2. Transformasi data pada orientasi layar *portrait.*

```
int w = vi.xres, h = vi.yres, w_aspect, crop;
int x, y, offset = 0;
uint32_t pixel32;

uint8_t *frame = (uint8_t *) malloc(640 * 480 * 3);

w_aspect = (w / 3) * 4;
crop = (h - w_aspect) / 2;
for (y = (w - 1); y >= 0; y--) {
    for (x = crop; x < (crop + w_aspect); x++) {
        pixel32 = ((uint32_t *) gr_framebuffer[0].data)[(480 *
                  x) + y];
        frame[3 * offset + 0] = pixel32;
        frame[3 * offset + 1] = pixel32 >> 8;
        frame[3 * offset + 2] = pixel32 >> 16;
        offset++;
    }
}
```

Kode 3.3. Transformasi data pada orientasi layar *landscape*.

3.2.3. *Socket Client Service*

Socket client service merupakan *service* untuk mengirimkan data hasil *screen capture* ke perangkat *VGA Adapter*. Data *screen capture* yang dikirim telah melewati proses transformasi data dan menghasilkan data gambar dengan *aspect ratio* 4:3. *Service* ini bertindak sebagai *socket client* yang bertugas mengirimkan data gambar ke perangkat *VGA Adapter* (*server*) secara kontinu dalam rentang waktu tertentu. Protokol yang digunakan untuk transmisi data yaitu tipe *stream socket* menggunakan *Transmission Control Protocol* (TCP).

Antara *client* dan *server* berkomunikasi pada *port* yang sama dan masing-masing memiliki alamat *Internet Protocol* (IP). Nomor *port* yang digunakan saat berkomunikasi harus dibuka terlebih dulu oleh *server*, baru kemudian *client* dapat memulai koneksi dengan menyertakan alamat IP *server*. Kode 3.4 menunjukkan inisialisasi *socket client* serta koneksi dengan *server* [13].

```
int sockfd, port;
struct sockaddr_in serv_addr;
struct hostent *server;

port = 2007;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0) {
     perror("ERROR opening socket");
}
server = gethostbyname("192.168.0.50");
if (server == NULL ) {
     perror("ERROR finding host");
}
bzero((char*)&serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char*)server->h_addr, (char*)&serv_addr.sin_addr.s_addr,
      server->h_length);
serv_addr.sin_port = htons(port);
if (connect(sockfd, (struct sockaddr*) &serv_addr,
     sizeof(serv_addr)) < 0) {
     perror("ERROR connecting socket");
}
```

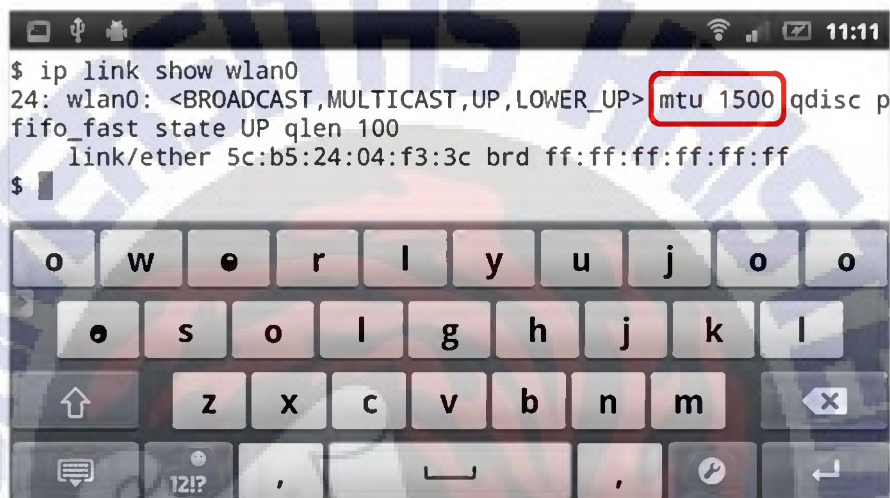
Kode 3.4. Inisialisasi *socket client* dan koneksi ke *server*.

Untuk membuka koneksi *socket* digunakan fungsi `socket()` dengan argumen pertama adalah *domain* komunikasi dari *socket* yang dibuka, argumen kedua adalah tipe *socket* dan argumen ketiga adalah protokol *socket*. Jika *socket* berhasil dibuka maka fungsi `socket()` akan mengembalikan *socket file descriptor* yang dapat digunakan dalam fungsi *socket* berikutnya.

Setelah *socket* berhasil dibuka langkah selanjutnya adalah melakukan koneksi ke *server* dengan menggunakan fungsi `connect()` dengan argumen pertama adalah *socket file descriptor*, argumen kedua adalah *struct* yang berisi informasi alamat *server* dan argumen ketiga adalah panjang *struct* dari argumen kedua. Jika koneksi dari *client* ke *server* berhasil dilakukan maka fungsi `connect()` akan mengembalikan nilai 0 dan *server* sudah siap menerima data.

Untuk pengiriman data *screen capture* digunakan fungsi `send()` dengan argumen pertama adalah *socket file descriptor*, argumen kedua adalah *buffer* data yang akan dikirimkan, argumen ketiga adalah panjang *buffer* data dan argumen keempat adalah tipe transmisi data. Fungsi `send()` akan mengembalikan jumlah *byte* dari data yang berhasil dikirimkan.

Pengiriman data menggunakan protokol TCP memiliki batas maksimal untuk jumlah data dalam sekali pengiriman yaitu 65535 *byte*. Pada kenyataannya jumlah *byte* yang dapat dikirimkan lebih kecil dari 65535 *byte* karena perangkat *Wi-Fi* internal Android memiliki *Maximum Transmission Unit* (MTU) yaitu 1500 *byte*. Gambar 3.9 menunjukkan nilai MTU perangkat *Wi-Fi* internal Android yang diambil lewat *command line* melalui aplikasi *Terminal Emulator*.



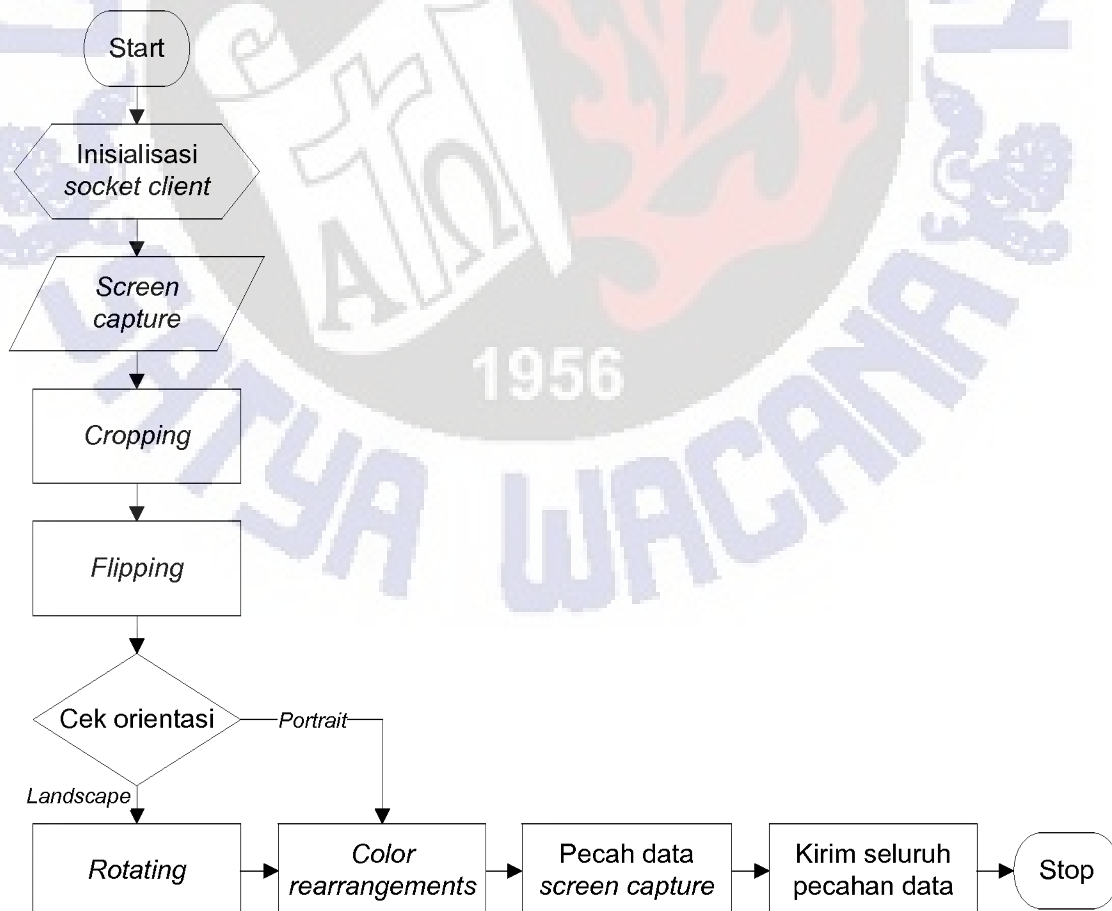
Gambar 3.9. Nilai MTU perangkat *Wi-Fi* Android.

Karena panjang data *screen capture* yang akan dikirim melebihi MTU perangkat *Wi-Fi* Android, maka pengirimannya dilakukan dengan cara memecah data menjadi bagian-bagian yang lebih kecil dari 1500 *byte*. Setelah data dipecah-pecah maka pengiriman dilakukan untuk setiap pecahan data sampai semua pecahan data terkirim. Kode 3.5 menunjukkan contoh pengiriman data *screen capture* berdimensi 640 x 480 piksel (tiap piksel terdiri dari 3 *byte* nilai warna RGB).

```
int x, y, n;
uint8_t *buff = (uint8_t *) malloc(640);

buff[0] = LANDSCAPE;
n = send(sockfd, buff, 1, 0);
if (n < 0) perror("ERROR writing to socket");
for (y = 0; y < 480 * 3; y++) {
     for (x = 0; x < 640; x++)
          buff[x] = frame[(y * 640) + x];
     n = send(sockfd, buff, 640, 0);
     if (n < 0) perror("ERROR writing to socket");
}
free(buff);
close(sockfd);
```

Kode 3.5. Pengiriman data *framebuffer* 640 x 480 piksel.

Seluruh proses yang ada dalam *service* mulai dari pengambilan data *framebuffer* (*screen capture*), tranformasi data (*cropping, flipping, rotating, color rearrangements*) sampai pengiriman data lewat *socket* ditunjukkan oleh diagram alir pada Gambar 3.10.



Gambar 3.10. Diagram alir *service* aplikasi *mobile* Android.

3.3. Aplikasi *Desktop*

Aplikasi *desktop* memiliki fungsi yang serupa dengan aplikasi *mobile* Android yaitu melakukan *capturing* pada layar secara kontinu dan mengirimkannya ke *VGA Adapter* melalui jaringan *Wi-Fi*. Aplikasi ini dibangun menggunakan bahasa pemrograman Java.

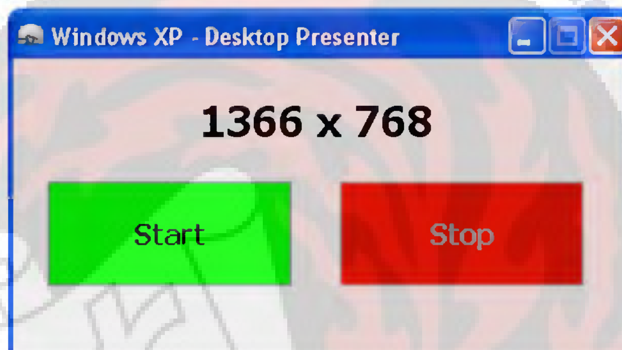
Untuk melakukan *capturing* pada layar digunakan kelas `java.awt.Robot` (Kode 3.6) dan untuk pengiriman data *screen capture* menggunakan protokol TCP dengan pemrograman *socket*. Pengiriman data *screen capture* juga dilakukan dengan teknik pemecahan data. Selain itu, aplikasi ini juga mampu mendeteksi resolusi layar untuk menentukan area pada layar yang akan di-*capture*. Jika resolusi layar lebih besar dari 800 x 600 piksel maka hasil *screen capture* diperkecil menjadi berdimensi 800 x 600 piksel menggunakan teknik *bilinear interpolation* (akan dibahas pada Kode 3.9, untuk Java tidak menggunakan *pointer* tetapi menggunakan *array*). Tujuannya adalah untuk mengurangi beban *payload* pada saat transmisi data sehingga *frame rate* yang dicapai bisa lebih tinggi.

```
Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
int screenWidth = dim.width, screenHeight = dim.height;
int aspectWidth = (screenHeight / 3) * 4;
int aspectHeight = screenHeight;

BufferedImage screencapture = null;
try {
     if (aspectWidth > 800)
          screencapture = new Robot()
               .createScreenCapture(new Rectangle(
                    (screenWidth - aspectWidth) / 2, 0,
                         aspectWidth, aspectHeight));
     if (aspectWidth == 800) {
          if (screenWidth == aspectWidth)
               screencapture = new Robot()
                    .createScreenCapture(new Rectangle(0,
                         0, aspectWidth, aspectHeight));
          else
               screencapture = new Robot()
                    .createScreenCapture(new Rectangle(
                         (screenWidth - aspectWidth) / 2,
                              0, aspectWidth, aspectHeight));
     }
} catch (AWTException e1) {
      lblError.setText("ERROR on capturing!");
}

int[] dataBuffInt = screencapture.getRGB(0, 0, aspectWidth,
          aspectHeight, null, 0, aspectWidth);
```

Kode 3.6. *Screen capture* pada aplikasi *desktop*.

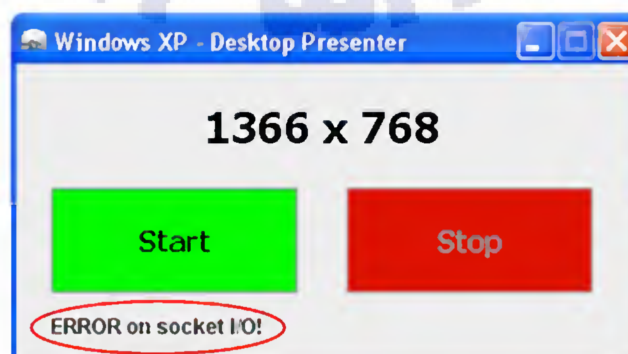
Untuk menjalankan aplikasi *desktop* ini, pengguna harus menghubungkan komputer atau *notebook* miliknya ke jaringan *Wi-Fi VGA Adapter* secara manual, lain halnya dengan aplikasi *mobile* Android yang mampu terhubung secara otomatis. Hal ini disebabkan karena Java tidak menyediakan API untuk akses langsung ke *wireless adapter* sehingga pengguna harus mencari SSID jaringan *Wi-Fi* dan memasukkan *password* secara manual.

Aplikasi *desktop* ini memiliki sebuah *graphical user interface* (GUI) sederhana dengan dua buah tombol yang masing-masing berfungsi untuk memulai dan menghentikan proses *screen capture* serta pengiriman data. Selain itu terdapat sebuah label yang berisi informasi resolusi layar yang terdeteksi. GUI ini menggunakan kelas-kelas yang terdapat pada `javax.swing`. Tampilan GUI pada aplikasi *desktop* dapat dilihat pada Gambar 3.11.



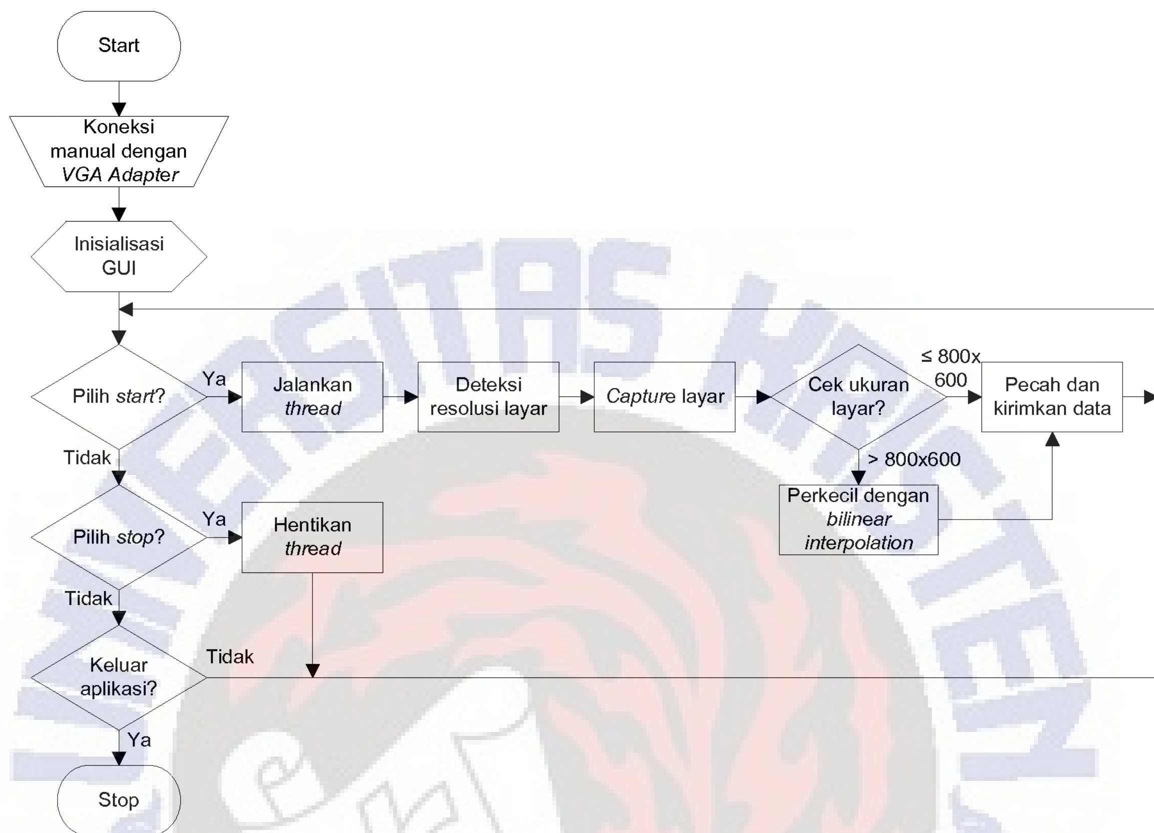
Gambar 3.11. Tampilan GUI aplikasi *desktop*.

Aplikasi *desktop* menggunakan sebuah *thread* yang digunakan untuk fungsi *screen capture* dan pengirimannya ke *VGA Adapter*. *Thread* ini mulai dijalankankan ketika tombol *start* ditekan, sebaliknya *thread* akan dihentikan ketika tombol *stop* ditekan. Aplikasi akan memberikan peringatan kesalahan (Gambar 3.12) jika tombol *start* ditekan pada saat komputer atau *notebook* tidak terhubung dengan *server* atau *VGA Adapter*.



Gambar 3.12. Peringatan kesalahan jika tidak terhubung dengan *server*.

Diagram alir aplikasi *desktop* dapat dilihat pada Gambar 3.13.



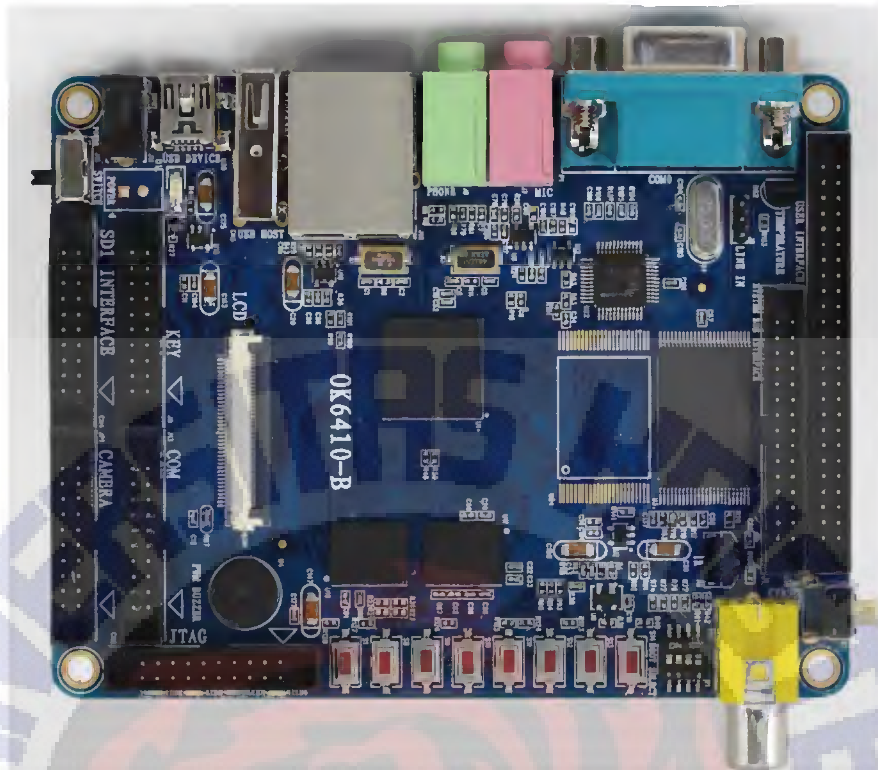
Gambar 3.13. Diagram alir aplikasi *desktop*.

3.4. *VGA Adapter*

VGA Adapter adalah perangkat penampil gambar dengan antarmuka VGA yang dapat menampilkan *slide-slide* presentasi dari aplikasi *mobile* Android. *Slide-slide* presentasi dikirimkan melalui jaringan nirkabel *Wi-Fi* dan ditampilkan pada proyektor digital. Perangkat keras yang terdapat dalam *VGA Adapter* yaitu modul mikroprosesor sebagai pengendali utama, modul *VGA Controller* sebagai penampil gambar dan *Wi-Fi Access Point* sebagai penyedia jaringan *Wi-Fi*.

3.4.1. Modul Mikroprosesor

Modul mikroprosesor menggunakan *development board* seri OK6410-B dari Forlinx yang dapat dilihat pada Gambar 3.14 [15, h.4]. Modul ini menggunakan mikroprosesor S3C6410 dari Samsung dengan arsitektur berbasis ARM1176JZF-S (ARM11). Mikroprosesor ini memiliki frekuensi kerja 532 MHz, RAM 256 MB dan dilengkapi dengan NAND Flash 2 GB.



Gambar 3.14. ***Development Board*** **OK6410-B.**

Modul mikroprosesor OK6410-B memiliki beberapa fitur yang digunakan dalam perancangan skripsi ini yaitu:

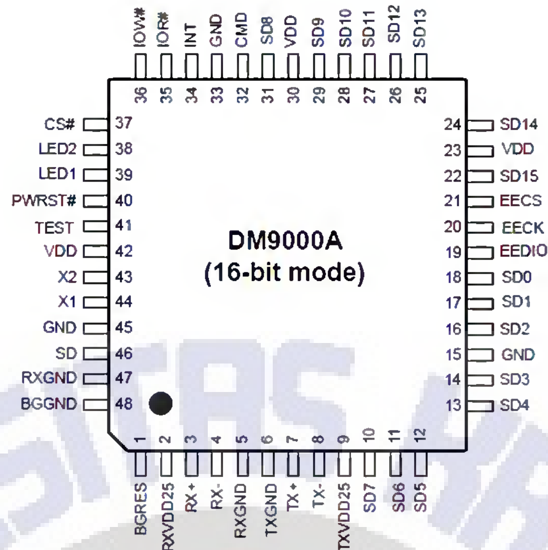
- *Display Interface*

  Konektor *display interface* digunakan untuk menampilkan gambar pada modul mikroprosesor OK6410-B. Pada perancangan skripsi ini konektor *display interface* dihubungkan dengan *VGA Controller*. Konektor ini menggunakan konektor FPC 40 pin yang dapat dilihat skema untainya pada Gambar 3.15 [15, h.12-13].

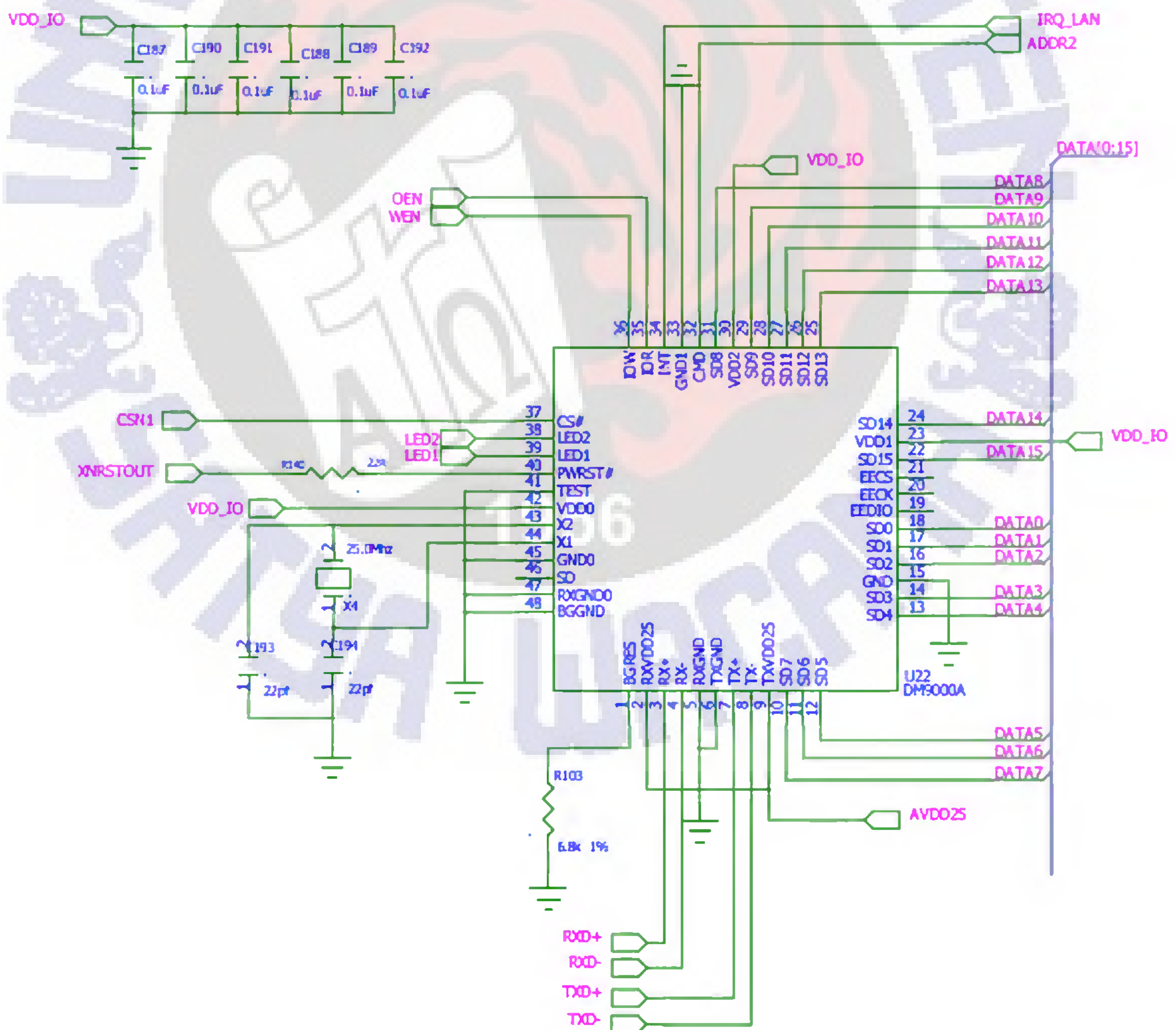
Gambar 3.15. Skema untai konektor *display interface* pada modul OK6410-B.

- *Ethernet Controller*

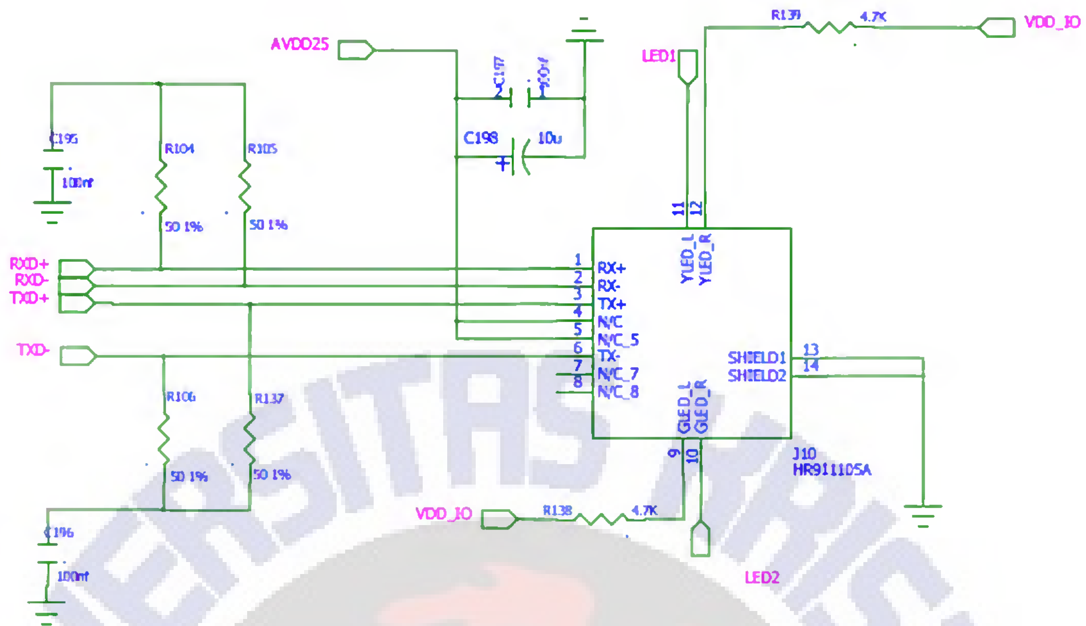
  Modul mikroprosesor OK6410-B menggunakan e*thernet controller* DM9000AE untuk mengimplementasikan *100M Ethernet* dan dilengkapi dengan konektor RJ45 [15, h.14]. Fitur ini digunakan dalam perancangan skripsi sebagai antarmuka dengan *Wi-Fi Access Point* melalui sebuah kabel *Unshielded Twisted Pair* (UTP) lewat konektor RJ45. Konfigurasi pin DM9000AE dapat dilihat pada Gambar 3.16 dan skema untainya pada modul mikroprosesor OK6410-B dapat dilihat pada Gambar 3.17 [15, h.15]. Sedangkan skema untai konektor RJ45 dapat dilihat pada Gambar 3.18 [15, h.16].



Gambar 3.16. Konfigurasi pin DM9000AE.



Gambar 3.17. Skema untai DM9000AE pada modul OK6410-B.



Gambar 3.18. Skema untai konektor RJ45 pada modul OK6410-B.

- Sistem Operasi *Embedded Linux* 3.0.1

Modul mikroprosesor OK6410-B didukung oleh sistem operasi *Embedded Linux* versi 3.0.1, yaitu sistem operasi berbasis Linux yang dioptimasi untuk perangkat *embedded.* Sistem operasi ini menyediakan *kernel* yang berisi *driver* untuk berbagai perangkat keras pada modul OK6410-B seperti pengontrol grafis dan *Ethernet controller* yang digunakan dalam perancangan skripsi ini. Sistem operasi *Embedded Linux* 3.0.1 juga bersifat *open source* sehingga berkas-berkas *filesystem* atau kode sumber *kernel* dapat dimodifikasi sesuai kebutuhan pada perancangan skripsi ini.

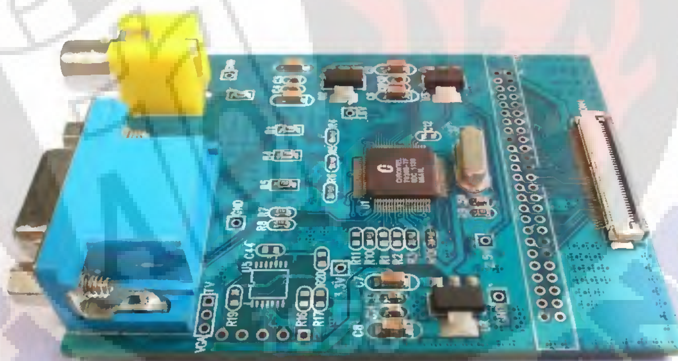
Pembuatan program pada sistem operasi *Embedded Linux* menggunakan *cross-compiler* dari GCC versi 4.2.2. Selain untuk pembuatan program, *cross-compiler* GCC 4.2.2 juga dapat digunakan untuk mengkompilasi kode sumber *kernel* yang digunakan pada *Embedded Linux* 3.0.1. Pada perancangan skripsi ini, kode sumber *kernel* dikompilasi ulang dengan beberapa modifikasi yaitu pengaturan resolusi *display* ke VGA 800 x 600 piksel serta menghilangkan *blinking cursor* di sudut kiri atas pada *display*.

Dengan menggunakan sistem operasi *Embedded Linux*, perangkat-perangkat keras yang terdapat pada modul OK6410-B dapat diakses oleh pengembang dengan antarmuka yang lebih mudah dipahami. Seluruh antarmuka *low-level* dengan

perangkat keras sudah didefinisikan di *kernel* dan *Embedded Linux* menyediakan API dalam bahasa pemrograman C/C++ untuk akses perangkat keras. Setiap perangkat keras didefinisikan di dalam berkas yang terdapat di dalam *filesystem* Linux, misalnya untuk akses ke *framebuffer* didefinisikan dalam berkas `/dev/fb0`. Sehingga untuk menggunakan suatu perangkat keras cukup dengan melakukan operasi baca atau tulis terhadap berkas *filesystem*-nya.

3.4.2. Modul *VGA Controller*

Modul *VGA Controller* berfungsi untuk menampilkan gambar keluaran dari modul mikroprosesor dengan antarmuka VGA. Dalam perancangan skripsi ini menggunakan modul *VGA Controller* dari Forlinx dengan resolusi keluaran 800 x 600 piksel dan kedalaman warna 16-bit (RGB565) yang dapat dilihat pada Gambar 3.19. Modul *VGA Controller* ini menggunakan IC *VGA Encoder* CH7026B-TF dari Chrontel dan dihubungkan ke konektor *display interface* pada modul mikroprosesor OK6410-B menggunakan kabel jenis FPC 40 pin.



Gambar 3.19. Modul *VGA Controller*.

3.4.3. *Wi-Fi Access Point*

Wi-Fi Access Point digunakan oleh perangkat *VGA Adapter* untuk komunikasi dan transmisi data dengan aplikasi *mobile* Android. *Access point* yang digunakan dalam perancangan skripsi ini yaitu TL-MR3020 dari TP-LINK yang dapat dilihat pada Gambar 3.20.

Gambar 3.20. ***Access point*** **TL-MR3020.**

Access point ini menyediakan jaringan *Wi-Fi* untuk aplikasi *mobile* Android dengan protokol otentifikasi WPA/WPA2. *Access point* ini juga bertindak sebagai *Dynamic Host Configuration Protocol* (DHCP) *server* yang menyediakan konfigurasi TCP/IP untuk seluruh sistem yang terhubung melalui LAN. Perangkat *VGA Adapter* terhubung dengan *access point* ini melalui kabel UTP lewat konektor RJ45, sedangkan aplikasi *mobile* Android akan terhubung melalui *Wi-Fi*.

3.4.4. Perangkat Lunak *VGA Adapter*

Perancangan perangkat lunak pada perangkat *VGA Adapter* meliputi *server socket* untuk menerima data *screen capture* dari aplikasi *mobile* Android dan *image viewer* untuk menampilkan gambar pada resolusi 800 x 600 piksel.

3.4.4.1. *Socket Server*

Socket server berfungsi untuk menerima data *screen capture* dari aplikasi *mobile* Android yang bertindak sebagai *client*. *Server* harus membuka *socket* terlebih dahulu sebelum *client* dapat mengirim data. Setelah *server* selesai membuka *socket*, *server* akan menunggu *client* memulai koneksi dan mengirim data. Kode 3.7 menunjukkan proses inisialisasi *socket* oleh *server* [13].

```
int sockfd, newsockfd, portno;
socklen_t clilen;
struct sockaddr_in serv_addr, cli_addr;

sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
      error("ERROR opening socket");
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
if(bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))
     < 0)
     error("ERROR on binding");
listen(sockfd, 5);
clilen = sizeof(cli_addr);
```

Kode 3.7. Inisialisasi *socket* pada *server.*

Proses inisialisasi *socket* pada *server* dan *client* hampir sama, yaitu untuk membuat *socket* pada *server* juga menggunakan fungsi `socket()` yang mengembalikan *socket file descriptor*. Setelah memanggil fungsi `socket()` langkah berikutnya adalah memanggil fungsi `bind()` yang berfungsi untuk mengikat *server* dengan *socket* yang sudah dibuat sebelumnya. Fungsi `bind()` memiliki tiga argumen yang pertama adalah *socket file descriptor,* argumen kedua berisi *struct* yang berisi informasi alamat *server* yang akan diikatkan dengan *socket* dan argumen ketiga adalah panjang *struct* pada argumen kedua.

Ketika *server* sudah terikat dengan *socket* maka *server* dapat memerintahkan *socket* untuk menunggu adanya koneksi dari *client* dengan memanggil fungsi `listen()`. Fungsi `listen()` memiliki argumen pertama yaitu *socket file descriptor* dan argumen kedua yaitu jumlah koneksi yang dapat menunggu ketika *server* sedang menangani sebuah koneksi.

Karena data yang dikirim oleh *client* dipecah terlebih dahulu, *server* akan melakukan penerimaan data sesuai jumlah pecahan data. Sebelum *client* dapat mengirim data*, client* harus melakukan koneksi dulu dengan *server* dan koneksi tersebut harus disetujui oleh *server*. Untuk menyetujui permintaan koneksi dari *client* digunakan fungsi `accept()` dengan argumen pertama adalah *socket file descriptor*, argumen kedua berisi *struct* yang berisi informasi alamat *client* dan argumen ketiga berisi panjang *struct* pada argumen kedua. Setelah *server* menyetujui koneksi dari *client, server* dapat mulai menerima data dari *client* menggunakan fungsi `recv()` dengan argumen pertama berupa *socket file*

descriptor, argumen kedua yaitu *buffer* data yang akan dikirimkan, argumen ketiga yaitu panjang *buffer* data dan argumen keempat yaitu tipe transmisi data. Fungsi `recv()` akan mengembalikan jumlah *byte* data yang diterima. Kode 3.8 menunjukkan contoh proses penerimaan data *screen capture* berdimensi 640 x 480 piksel dari *client*.

```
int n, offset = 0, newsockfd;
buffer = (unsigned char *) malloc(640);
input = (unsigned char *) malloc(640 * 480 * 3);

newsockfd = accept(sockfd,(struct sockaddr*)&cli_addr,&clilen);
if (newsockfd < 0) error("ERROR on accept");
while (1) {
     n = recv(newsockfd, buffer, 640, 0);
     for (x = 0; x < n; x++)
          input[offset + x] = buffer[x];
     offset += n;
     if (offset == (640 * 480 * 3)) break;
}
```

Kode 3.8. Penerimaan data *framebuffer* 640 x 480 piksel.

3.4.4.2. *Image Viewer*

Image viewer merupakan bagian untuk menampilkan data *screen capture* yang dikirim oleh *client* pada keluaran VGA (800 x 600 piksel). Karena data yang dikirim *client* memiliki resolusi yang lebih kecil dari 800 x 600 piksel maka perlu dilakukan proses *scaling* dengan teknik *bilinear interpolation*. Teknik ini dipilih karena tidak membutuhkan proses komputasi yang rumit serta kualitas hasil *scaling* tetap terjaga atau tidak menimbulkan efek pecah pada gambar. Kode 3.9 menunjukkan contoh proses *scaling* data *screen capture* dengan dimensi 640 x 480 piksel menjadi 800 x 600 piksel dengan teknik *bilinear interpolation* [12].

```
unsigned int x, y;
unsigned int wStepFixed16b, hStepFixed16b, wCoef=0, hCoef=0;
unsigned char *pixel1, *pixel2, *pixel3, *pixel4;
unsigned int hc1, hc2, wc1, wc2, offsetX, offsetY;
unsigned int red, green, blue, offset=0;
int source=640, source=480, targetW=800, targetH=600;

wStepFixed16b = ((sourceW - 1) << 16) / (targetW - 1);
hStepFixed16b = ((sourceH - 1) << 16) / (targetH - 1);
for (y = 0; y < targetH; y++) {
    offsetY = (hCoef >> 16);
    hc2 = (hCoef >> 9) & 127;
    hc1 = 128 - hc2;
    wCoef = 0;
    for (x = 0; x < targetW; x++) {
        offsetX = (wCoef >> 16);
        wc2 = (wCoef >> 9) & 127;
        wc1 = 128 - wc2;
        pixel1 = &input[3*(offsetY * sourceW + offsetX)];
        pixel2 = &input[3*((offsetY+1) * sourceW + offsetX)];
        pixel3 = &input[3*(offsetY * sourceW + offset + 1)];
        pixel4 = &input[3*((offsetY+1) * sourceW + offsetX +
                 1)];
        red = ((*(pixel1+0)*hc1 + *(pixel2+0)*hc2)* wc1 +
               (*(pixel3+0)*hc1 + *(pixel4+0)*hc2)*wc2)>> 14;
        green = ((*(pixel1+1)*hc1 + *(pixel2+1)*hc2)*wc1 +
                 (*(pixel3+1)*hc1 + *(pixel4+1)*hc2)*wc2)>> 14;
        blue = ((*(pixel1+2)*hc1 + *(pixel2+2)*hc2)*wc1 +
                (*(pixel3+2)*hc1 + *(pixel4+2)*hc2)*wc2)>>14;
        output[(3 * offset) + 0] = red;
        output[(3 * offset) + 1] = green;
        output[(3 * offset) + 2] = blue;
        offset++;
        wCoef += wStepFixed16b;
    }
    hCoef += hStepFixed16b;
}
```

Kode 3.9. *Scaling* dengan teknik *bilinear interpolation*.

Untuk menampilkan gambar pada keluaran modul OK6410-B dapat dilakukan dengan membuka berkas *framebuffer* `fb0` dan memetakan memori untuk keperluan baca dan tulis. Modul OK6410-B hanya mendukung tampilan grafis dengan kedalaman warna maksimal 16-bit sehingga panjang memori yang dipetakan merupakan kelipatan 16-bit. Inisialisasi tampilan grafis pada modul OK6410-B dapat dilihat pada Kode 3.10.

```
int display_init() {
     int fb_fd = -1;
     __u32 screensize;

     fb_fd = open("/dev/fb0", O_RDWR);
     if (fb_fd < 0) {
          printf(" open FB ERR\n");
          return 0;
     }

     screensize = SCR_COLS * SCR_ROWS * 16 / 8;
     fb_buf = (char *) mmap(0, screensize, PROT_READ|PROT_WRITE,
                  MAP_SHARED, fb_fd, 0);
     if ((int) fb_buf == -1) {
          printf("Error: failed to map framebuffer\n");
          close(fb_fd);
          return -1;
     }
}
```

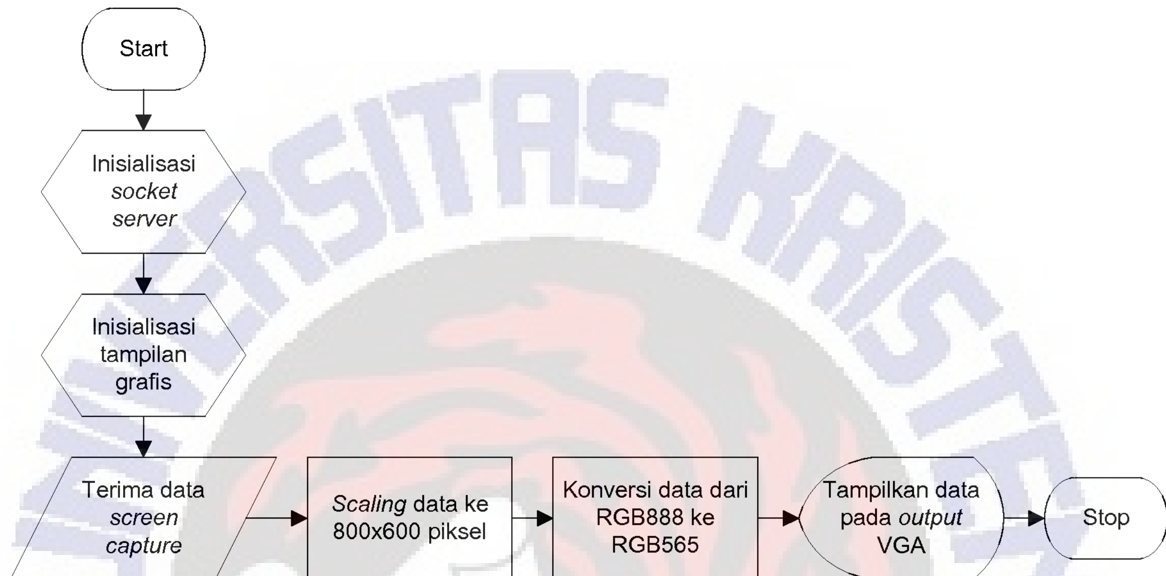
Kode 3.10. Inisialisasi tampilan grafis modul OK6410-B.

Karena modul OK6410-B hanya mendukung tampilan grafis dengan kedalaman warna maksimal 16-bit atau RGB565, sedangkan data *screen capture* memiliki kedalaman warna 24-bit atau RGB888, maka sebelum data *screen capture* ditampilkan data tersebut harus dikonversi menjadi format RGB565. Hal ini dilakukan dengan mengambil bit-bit MSB pada tiap kanal warna yaitu 5-bit merah, 6-bit hijau dan 5-bit biru kemudian digabungkan menjadi konfigurasi RGB565. Fungsi untuk konversi RGB888 menjadi RGB565 dan untuk menampilkan data *screen capture* pada keluaran VGA modul OK6410-B ditunjukkan pada Kode 3.11.

```
void show_rgb565_img(__u8 *scr, __u8 *pixel) {
    __u16 x, y;
    __u16 *fb_buf = (__u16 *) scr;
    for (y = 0; y < SCR_ROWS; y++) {
         for (x = 0; x < SCR_COLS; x++) {
              fb_buf[x] = ((((__u8)*pixel)>>3)<<11) | ((((__u8)
                             *(pixel+1))>>2)<<5) | (((__u8)*(pixel
                             +2))>>3);
              pixel += 3;
         }
         fb_buf += SCR_COLS;
    }
}
```

Kode 3.11. Konversi RGB888 ke RGB565 dan tampilan pada *output* VGA.

Diagram alir pada Gambar 3.21 menunjukkan seluruh proses di dalam perangkat *VGA Adapter* mulai dari penerimaan data *screen capture* dari aplikasi *mobile* Android sampai menampilkan data *screen capture* pada keluaran VGA modul mikroprosesor OK6410-B.



Gambar 3.21. Diagram alir perangkat *VGA Adapter*.