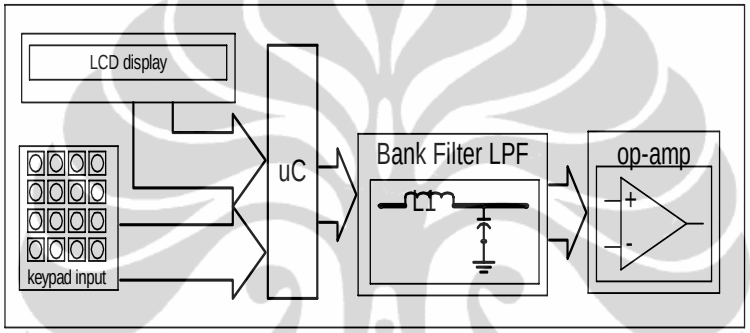
BAB III
METODE PENELITIAN

3.1 Perancangan *PWM* Generator untuk Pembangkitan Gelombang Sinus.

Pada Bab Pendahuluan telah dijelaskan bahwa penelitian ini dibagi menjadi 2 buah bagian, yang pertama perancangan generator *PWM* untuk pembangkitan gelombang sinus, yang kedua perancangan generator *8-PSK*. Bagan perancangan *PWM* sebagai dasar pembentukan sinus terlihat seperti Gambar 3.1.



Gambar 3.1 Blok diagram pembangkitan sinus

Pada perancangan generator *PWM* hal pertama yang dilakukan adalah membuat ilustrasi / sintesis pembentukan sinyal *PWM* oleh rangkaian *comparator* dengan input sinyal sinus dan sinyal segitiga. Ilustrasi ini dapat dilihat pada Lampiran 3.1 dengan nama file **Ilustrasi Perhitungan Lebar Pulsa PWM.exe**

Di dalam file ilustrasi tersebut terlihat bahwa sinyal sinus dibandingkan amplitudonya dengan sinyal segitiga dengan frekuensi lebih tinggi (ada 3 macam pilihan *sampling* yaitu 40 kali, 50 kali, dan 100 kali). Jika amplitudo sinyal sinus lebih tinggi daripada amplitudo sinyal segitiga maka terbentuk *PWM* periode *ON* atau *HIGH*. Jika amplitudo sinyal sinus lebih rendah daripada sinyal segitiga maka terbentuk *PWM* periode *OFF* atau *LOW*. Demikian seterusnya sehingga terbentuk sinyal *PWM* untuk periode 1 gelombang sinus.

Setelah terbentuk sinyal *PWM* untuk periode 1 gelombang sinus dilanjutkan dengan penghitungan lebar periode *ON* dan *OFF* tiap-tiap sinyal *PWM*-nya. Data periode *ON* dan *OFF* sinyal *PWM* ini selanjutnya digunakan

sebagai dasar pembangkitan sinyal *PWM* yang serupa dengan ilustrasi. Dari ilustrasi ini untuk sinyal sinus yang di-*compare*-kan (menggunakan rangkaian *comparator*) dengan 100 sinyal segitiga, didapat data seperti Tabel 3.1.

Tabel 3.1 adalah didapat dari asumsi bahwa 1 sinyal sinus = 800 piksel =100 sinyal segitiga, sehingga bila periode sinyal sinusnya diasumsikan adalah 800 ms (sesuai hukum f =1/ T, yang berarti frekuensi sinusnya adalah 1,25 Hz) maka periode *PWM* pertama adalah 3 ms untuk *Off* dan 5 ms untuk *On*. Begitu seterusnya untuk periode yang lainnya.

Tabel 3.1 Perhitungan periode *PWM*.

Sinyal	Periode		sinyal	Periode		sinyal	Periode		sinyal	Periode	
Ke	***Off***	***On***	ke	***Off***	***On***	ke	***Off***	***On***	ke	***Off***	***On***
1	3	5	**26**	1	7	**51**	4	4	**76**	7	1
2	3	5	**27**	1	7	**52**	5	3	**77**	7	1
3	3	5	**28**	1	7	**53**	5	3	**78**	7	1
4	3	5	**29**	1	7	**54**	5	3	**79**	7	1
5	3	5	**30**	1	7	**55**	5	3	**80**	7	1
6	3	5	**31**	1	7	**56**	5	3	**81**	7	1
7	3	5	**32**	1	7	**57**	5	3	**82**	7	1
8	3	5	**33**	1	7	**58**	5	3	**83**	7	1
9	3	5	**34**	1	7	**59**	5	3	**84**	7	1
10	3	5	**35**	1	7	**60**	5	3	**85**	7	1
11	3	5	**36**	1	7	**61**	5	3	**86**	7	1
12	3	5	**37**	1	7	**62**	5	2	**87**	7	1
13	2	7	**38**	1	6	**63**	7	1	**88**	7	1
14	1	7	**39**	2	6	**64**	7	1	**89**	6	3
15	1	7	**40**	3	5	**65**	7	1	**90**	5	3
16	1	7	**41**	3	5	**66**	7	1	**91**	5	3
17	1	7	**42**	3	5	**67**	7	1	**92**	5	3
18	1	7	**43**	3	5	**68**	7	1	**93**	5	3
19	1	7	**44**	3	5	**69**	7	1	**94**	5	3
20	1	7	**45**	3	5	**70**	7	1	**95**	5	3
21	1	7	**46**	3	5	**71**	7	1	**96**	5	3
22	1	7	**47**	3	5	**72**	7	1	**97**	5	3
23	1	7	**48**	3	5	**73**	7	1	**98**	5	3
24	1	7	**49**	3	5	**74**	7	1	**99**	5	3
25	1	7	**50**	4	4	**75**	7	1	**100**	4	4

Dengan logika yang mirip, bila kita ingin sinyal sinusnya mempunyai frekuensi 100 Hz, maka perhitungannya adalah sebagai berikut:

Untuk frekuensi sinus = 100 Hz yang berarti periode 10.000 mikrodetik = 800 piksel, maka tabelnya menjadi seperti pada Table 3.2. Dari Tabel 3.2 periode (T) sinyal sinusnya adalah 10.000 uS (yang berarti frekuensi (f) sinusnya adalah 100 Hz) maka periode *PWM* pertama adalah 38 uS untuk *Off* dan 68 uS untuk *On* (nilai ini diperoleh dari mengalikan nilai periode dengan (10.000 uS/800 piksel)). Begitu seterusnya untuk periode yang lainnya. Hal yang sama juga diterapkan untuk frekuensi 80 Hz, 50 Hz, dan 40 Hz.

Table 3.2 Periode *PWM* untuk sinus 100 Hz

Sinyal	T(uS)		sinyal	T(uS)		sinyal	T(uS)		sinyal	T(uS)	
Ke	***Off***	***On***	ke	***Off***	***On***	ke	***Off***	***On***	ke	***Off***	***On***
1	38	63	**26**	13	88	**51**	50	50	**76**	88	13
2	38	63	**27**	13	88	**52**	63	38	**77**	88	13
3	38	63	**28**	13	88	**53**	63	38	**78**	88	13
4	38	63	**29**	13	88	**54**	63	38	**79**	88	13
5	38	63	**30**	13	88	**55**	63	38	**80**	88	13
6	38	63	**31**	13	88	**56**	63	38	**81**	88	13
7	38	63	**32**	13	88	**57**	63	38	**82**	88	13
8	38	63	**33**	13	88	**58**	63	38	**83**	88	13
9	38	63	**34**	13	88	**59**	63	38	**84**	88	13
10	38	63	**35**	13	88	**60**	63	38	**85**	88	13
11	38	63	**36**	13	88	**61**	63	38	**86**	88	13
12	38	63	**37**	13	88	**62**	63	25	**87**	88	13
13	25	88	**38**	13	75	**63**	88	13	**88**	88	13
14	13	88	**39**	25	75	**64**	88	13	**89**	75	38
15	13	88	**40**	38	63	**65**	88	13	**90**	63	38
16	13	88	**41**	38	63	**66**	88	13	**91**	63	38
17	13	88	**42**	38	63	**67**	88	13	**92**	63	38
18	13	88	**43**	38	63	**68**	88	13	**93**	63	38
19	13	88	**44**	38	63	**69**	88	13	**94**	63	38
20	13	88	**45**	38	63	**70**	88	13	**95**	63	38
21	13	88	**46**	38	63	**71**	88	13	**96**	63	38
22	13	88	**47**	38	63	**72**	88	13	**97**	63	38
23	13	88	**48**	38	63	**73**	88	13	**98**	63	38
24	13	88	**49**	38	63	**74**	88	13	**99**	63	38
25	13	88	**50**	50	50	**75**	88	13	**100**	50	50

Langkah selanjutnya adalah menggunakan daftar tabel diatas sebagai data referensi pembangkitan sinyal *PWM* melalui implementasi *software*.

Karena sinyal *PWM* masih berbentuk digital maka langkah selanjutnya untuk mengubah menjadi sinus, yaitu dengan memfilternya dengan filter *lowpass*. Adapun penentuan nilai komponen filter yang digunakan, dilakukan dengan perhitungan yang telah dijelaskan pada Bab II mengenai filter *lowpass*, dan dilanjutkan dengan simulasi.

Dalam perancangan filter di gunakan persamaan pada Bab 2, misalkan dirancang rangkaian filter yang berfungsi memfilter frekuensi *carrier* 8 kHz, maka digunakan nilai L=1 mH dan C= 390 nF dan bila dirancang rangkaian filter yang berfungsi memfilter frekuensi *carrier* 80 Hz, maka digunakan nilai L=100 mH dan C= 39 uF.

Tabel 3.3 Nilai LC untuk f1 = 8 kHz dan f2 = 80 Hz

No	Frekuensi Hz	Nilai pokok L	C
1	8000	1 mH	390 nF
2	80	100 mH	39 uF

Kemudian untuk memulai implementasi pada mikrokontroler hal yang dilakukan adalah perancangan algoritma dan *flowchart*-nya. Dari algoritma ini kemudian diturunkan ke dalam bahasa pemrograman.

Setelah melakukan simulasi dengan filter *lowpass* LC, ditemukan bahwa hasil pemfilteran dengan menggunakan filter L=1 mH dan C= 390 nF hasilnya masih ditemukan derau, sedangkan menggunakan filter L=100 mH dan C= 39 uF hasilnya berupa sinyal segitiga. Untuk itu perlu dilakukan modifikasi yaitu menggabungkan rangkaian *filter lowpass* LC dengan filter *lowpass* orde satu yaitu filter RC.

Nilai RC yang akan dibuat sebagai rangkaian *filter lowpass first order* harus memenuhi persamaan berikut ini:

$$R1\,C1 = \frac{1}{(2\pi fL)} \qquad \text{(P3.1)}$$

Persamaan 3.1 diatas terlihat ada 3 variabel dimana f adalah frekuensi *carrier*, R adalah nilai resistansi dan C adalah nilai kapasitansi, sebagai contohnya jika

dilakukan perancangan *filter* yang memfilter suatu sinyal dengan frekuensi *carrier* 8 kHz, maka penentuan nilai RC nya adalah sebagai berikut:

$$R1\ C1 = \frac{1}{(2\ \pi\ f1\ )}$$

Anggap nilai kapasitansi yang dipakai adalah R1 = 1600 Ohm, sehingga

$$C1 = \frac{1}{(2\ \pi\ 8000\ )\ 1600}$$

$$C1 = 12\ nF$$

Maka pasangan nilai RC yang dipakai untuk memfilter frekuensi *carrier* 8 khz adalah R1 = 1600 Ohm dan C1 = 12 nF .

Untuk frekuensi *carrier* 80 Hz, maka penentuan nilai RC nya adalah sebagai berikut:

$$R2\ C2 = \frac{1}{(2\ \pi\ f2\ )} \qquad \text{(P3.2)}$$

Di karenakan fc1 = 100 fc2 maka

$$R2\ C2 = \frac{1}{(2\ \pi\ f2\ )}$$

$$f2 = \frac{1}{(2\ \pi\ R2\ C2\ )}$$

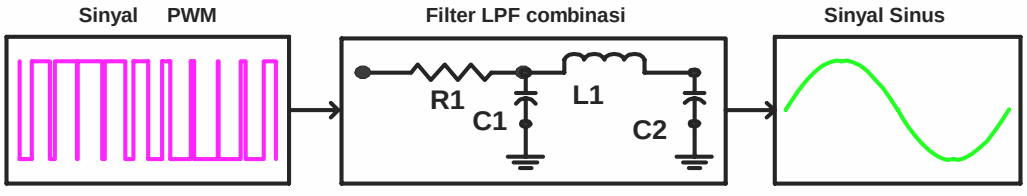
$$f2 = \frac{1}{100(2\ \pi\ R1\ C1\ )}$$

sehingga,

$$f2 = \frac{1}{(2\ \pi\ 10R1\ 10C1\ )}$$

dengan demikian diperoleh bahwa R2 = 10 R1 = 16 kOhm, dan C2 = 10C1= 120 nF, maka pasangan nilai RC yang dipakai untuk memfilter frekuensi *carier* 80 Hz adalah R2= 16 kOhm dan C2 = 120 nF.

Setelah mendapatkan nilainya maka filter RC digabung dengan filter LC. Skema filter kombinasi yang dimaksud adalah pada Gambar 3.2.



Gambar 3.2 Filter kombinasi LC dengan RC

Nilai L1 dan C2 diambil dari Tabel 3.3, sedangkan nilai R1 dan C1 didapat dari Persamaan 3.1 dengan menyesuaikan frekuensi *carrier*nya masing-masing. Tabel 3.4 di bawah ini menunjukkan nilai filter gabungan *first order* dan *second order*.

Tabel 3.4 *Lowpass filter* gabungan antara *first order* dan *second order* sebagai filter *PWM*.

	Frekuensi	*Second Order*		*First Order*	
No	Hz	L	C	R	C
1	8000	1 mH	390 nF	1600 Ohm	12 nF
2	80	100 mH	39 uF	16 kOhm	120 nF

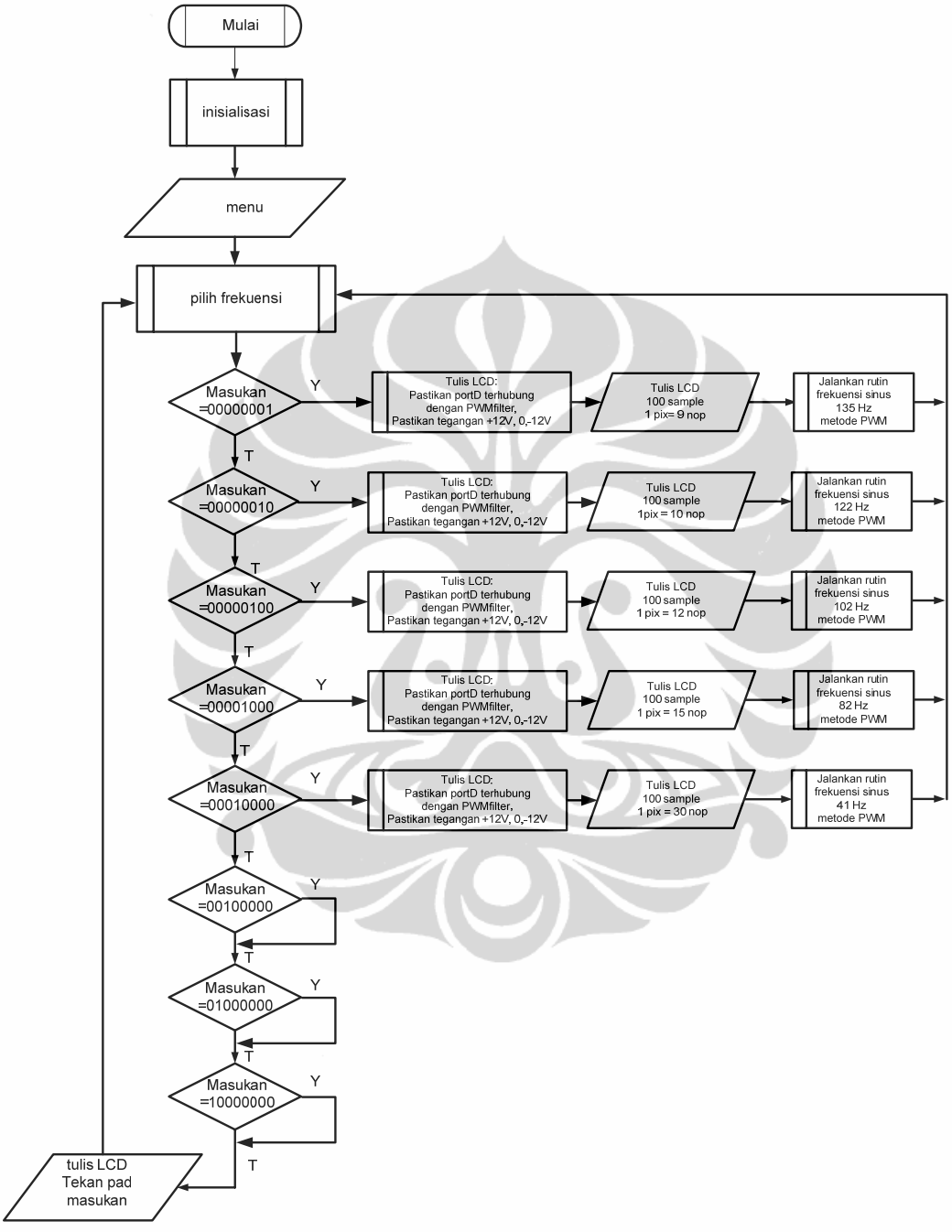
Untuk memperjelas bahwa, dengan menggunakan filter *lowpass second order* (filter LC) yang dikombinasikan dengan filter *lowpass firt order* (filter RC) menghasilkan sinyal yang lebih baik, maka dapat dilihat pada file simulasi yaitu **filter_frequency_fc8k_10R_10C_L_C.ms9** (lampiran 3.2), **filter_frequency_fc8k_R_C_100L_100C.ms9** (lampiran 3.3).

Pada laporan ini yang dibuat adalah 4 buah *PWM* yaitu 13500 Hz, 12200 Hz, 10200 Hz, 8300 Hz dan 4100 Hz masing – masing membawa sinyal informasi sinus yaitu 135 Hz, 122 Hz, 102Hz, 83 Hz dan 41 Hz karena hal ini dianggap cukup mewakili *sinus generator* yang dibuat. Jika simulasi dan skematik sudah dibuat maka selanjutnya adalah perancangan *hardware* dan *software*-nya.

Pada perancangannya hal awal yang dilakukan dalam merancang *software PWM* adalah membangkitkan sinyal *high low* dengan periode *on off* di buat sesuai dengan tabel. Program ini merupakan pokok pikiran yang digunakan guna membangun *software* pembangkit *PWM*. Adapun *flowchart*-nya adalah seperti ditunjukkan Gambar 3.3.

Deskripsi *flowchart* : pada awalnya mikrokontroler melakukan inisialisasi kemudian memberikan pilihan menu frekuensi yang tersedia dan memberikan kesempatan pengguna untuk memilih salah satu. Jika tidak ada masukan maka mikrokontroler akan menampilkan menu terus (*looping*), jika ada masukan maka

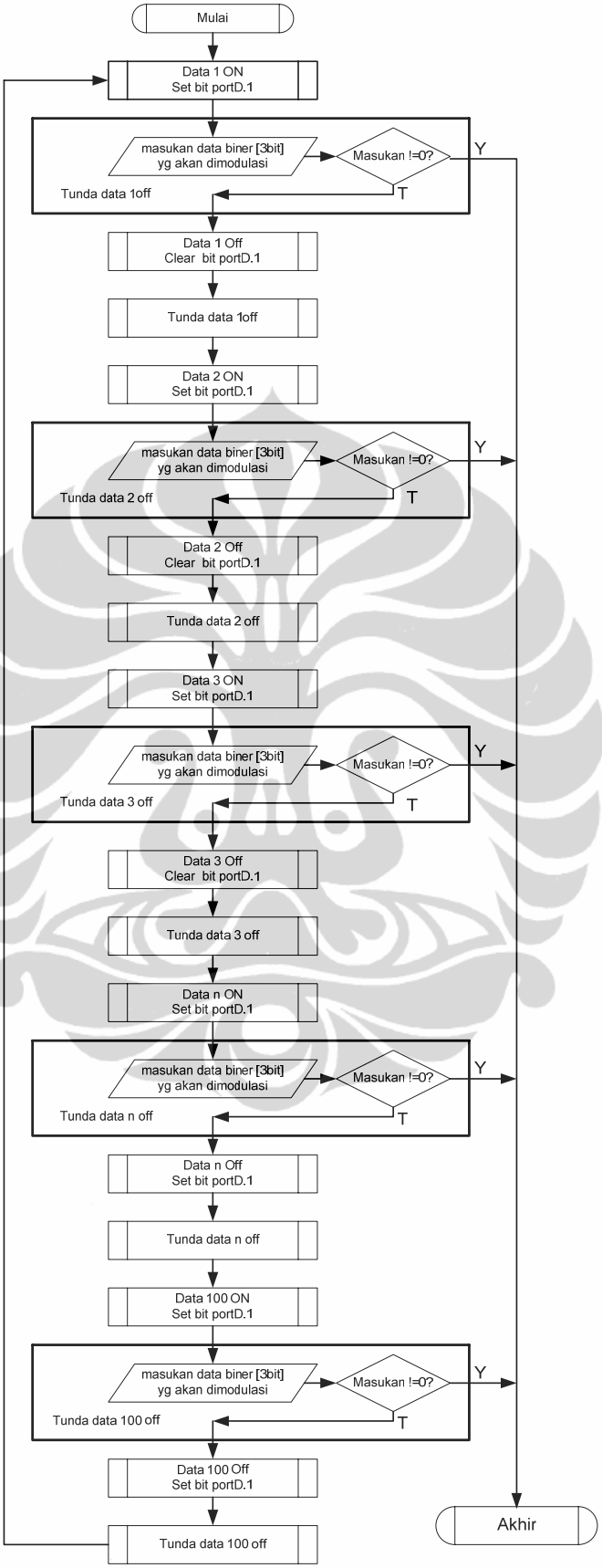
mikrokontroler akan menjalankan rutin program pembangkitan *PWM* sesuai dengan masukan yang diberikan.



Gambar 3.3 Flowchart pembangkitan *PWM*

Pada Gambar 3.3 *flowchart* pembangkitan *PWM* tampak bahwa Rutin jalankan rutin frekuensi sinus 135 Hz metode *PWM* (termasuk juga 122 Hz,

102Hz, 83 Hz dan 41 Hz) tidak dijelaskan secara rinci, Gambar 3.4 menjelaskan lebih rinci pembangkitan *PWM* tersebut.



Gambar 3.4 *Flowchart* pembangkitan sinus 135 Hz metode *PWM*

Deskripsi *flowchart* : mikrokontroler melakukan *toggle* dari *LOW* ke *HIGH* berulang–ulang sesuai periode *ON OFF*, saat tundaan lama maka mikrokontroler akan melakukan pengambilan data dari *pad input* untuk melakukan pengecekan apakah ada masukan atau tidak. Jika ada masukan maka keluar dari *loop ON OFF*, jika tidak ada masukan tetap *looping* dalam rutin *PWM*.

Yang membedakan sinyal *PWM* untuk sinus 135 Hz, 122 Hz, 102Hz, 83 Hz dan 41 Hz terletak pada nilai *rasio* tundaannya: untuk 135 Hz *rasio* tundaannya 9 *nop*, 122 Hz (10 *nop*), 102Hz (12 *nop*), 83 Hz (15 *nop*) dan 41 Hz (30 *nop*). *Nop* (*No Operation*) merupakan perintah bahasa *assembler* dengan waktu eksekusi 1 siklus mesin.

3.2 Perancangan modulator *8-PSK*

Pada modulasi *8-PSK* sinyal informasi terletak pada fasa sinus. Tiap fasa mengindikasikan suatu data *biner* tertentu. Fasa dalam kontek ini adalah sudut start dimana sinyal sinus mulai, sehingga dalam perancangan modulator *8-PSK* dibutuhkan rangkaian penggeser fasa, selain rangkaian penggeser fasa, juga di butuhkan pendeteksi *zero cross* (*hardware* dan *software*), generator 3-bit (*software*) juga perancangan pensaklaran (*hardware* dan *software*). Deskripsi tiap-tiap sub-bagian ini adalah sebagai berikut:

1. Perancangan rangkaian penggeser fasa. Fasa –fasa sinus yang digeser mewakili nilai *biner* tertentu. Fasa-fasa yang digeser adalah penggeseran fasa 0˚, 30˚, 45˚, 60˚, 90˚, 120˚, 135˚, 150˚, 180˚, 210˚, 225˚, 240˚, 270˚, 300˚, 315˚, 330˚.
2. Perancangan *zero cross* (terdiri *hardware* dan *software*), yang berfungsi mendeteksi titik sentuh sinyal sinus dengan sumbu 0. Dengan kata lain mencari tahu kapan mikrokontroler harus melakukan pemicuan/ pensaklaran.
3. Perancangan generator 3-bit (*software*), yang berfungsi membangkitkan data *biner* 3 bit sebagai *source* / sumber yang akan di modulasi.

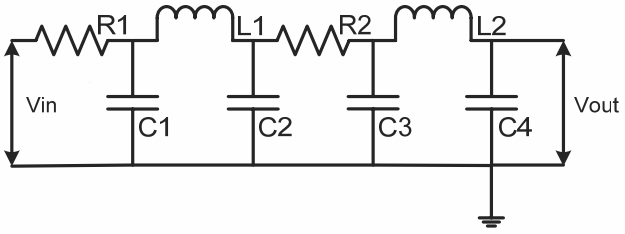
4. Perancangan pensaklaran (terdiri dari *hardware* dan *software*), yang berfungsi melakukan pensaklaran sinyal sinus sesuai *input* dari generator 3-bit saat *zero cross* terjadi.

3.2.1 Perancangan *Phase shifter* / penggeser fasa

Bila perancangan generator sinus sudah jadi maka langkah selanjutnya adalah perancangan generator *8-PSK.* Metode yang dilakukan adalah dengan membuat simulasi penggeseran fasa melalui program bantu simulator Multisim versi 8, adapun fasa-fasa yang digeser adalah penggeseran fasa 0˚, 30˚, 45˚, 60˚, 90˚, 120˚, 135˚, 150˚, 180˚, 210˚, 225˚, 240˚, 270˚, 300˚, 315˚, 330˚.

Dengan bekal uraian dan simulasi pada Bab 2 tentang pergeseran fasa maka dibentuk semua rangkaian penggeseran fasanya. Hal ini terlihat jelas pada bab file **shifter1.SCH** (lampiran 3.4). Seperti yang di perlihatkan pada **file non_invshift30.ms8** (lampiran 3.5), **non_invshiftB45.ms8** (lampiran 3.6), dan **non_invshiftB60.ms8** (lampiran 3.7), dalam perancangannya diusahakan agar hanya dengan merubah-ubah nilai R maka didapat pergeseran fasa yang diinginkan (tanpa harus melakukan perubahan pada nilai komponen lainnya seperti L maupun C).

Gambar 3.5 merupakan rangkaian skematik penggeser fasa, dengan rangkaian ini pergeseran fasa 30˚, 45˚, dan 60˚ dapat dibentuk. Uraian mengenai penurunan persamaan rangkaian ini yang menunjukkan bahwa variabel R berpengaruh terhadap fasa dapat dilihat pada Lampiran 3.9



Gambar 3.5 Rangkaian skematik penggeser fasa

Fasa-fasa yang digeser (fasa 0˚, 45˚, 90˚, 135˚, 180˚, 225˚, 270˚, 315˚) dihubungkan dengan *input IC analog switch* yang terdapat pada file **switch1.SCH** (lampiran 3.8).

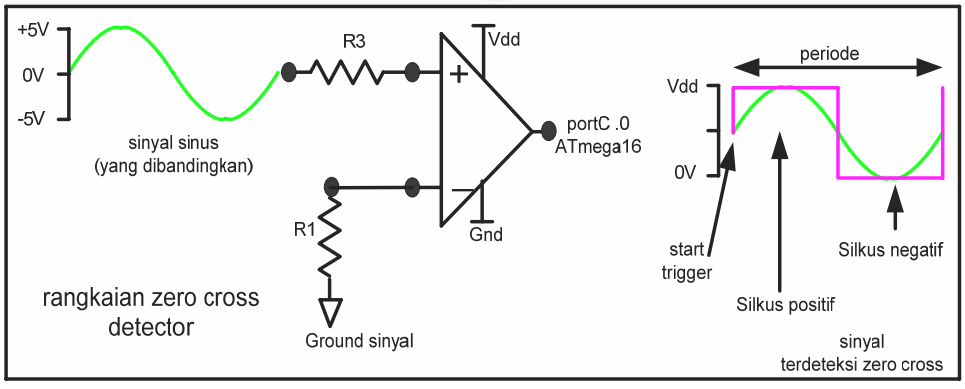
IC analog switch terdiri dari 3 bagian penting yaitu *input analog*, *output analog* dan *switch* kontrol. *Input analog* terkoneksi dengan fasa-fasa modulasi. *Switch* kontrol secara tak langsung terhubung dengan mikrokontroler melalui *comparator*. *Output* IC ini terkoneksi pada satu titik yang sekaligus merupakan titik pengamatan modulasi *8-PSK*. File **switch1.SCH** (Lampiran 3.8) merupakan rangkaian pensaklaran (*switching*) yang berfungsi sebagai rangkaian yang akan men-*switch* fasa mana saja yang ON yang bersesuaian dengan informasi yang akan dimasukan. Jika kedua hal diatas telah dibuat simulasi dan skematiknya maka selanjutnya adalah perancangan *hardware* yang didasarkan atas skematik.

3.2.2 Perancangan *Zero Cross*

Zero cross merupakan suatu *hardware* yang berfungsi untuk mendeteksi adanya titik sentuh sinyal sinus dengan tegangan nol. Titik sentuh ini berarti sebagi fasa nol. Ada 2 titik sentuh sinyal sinus dengan sumbu 0:

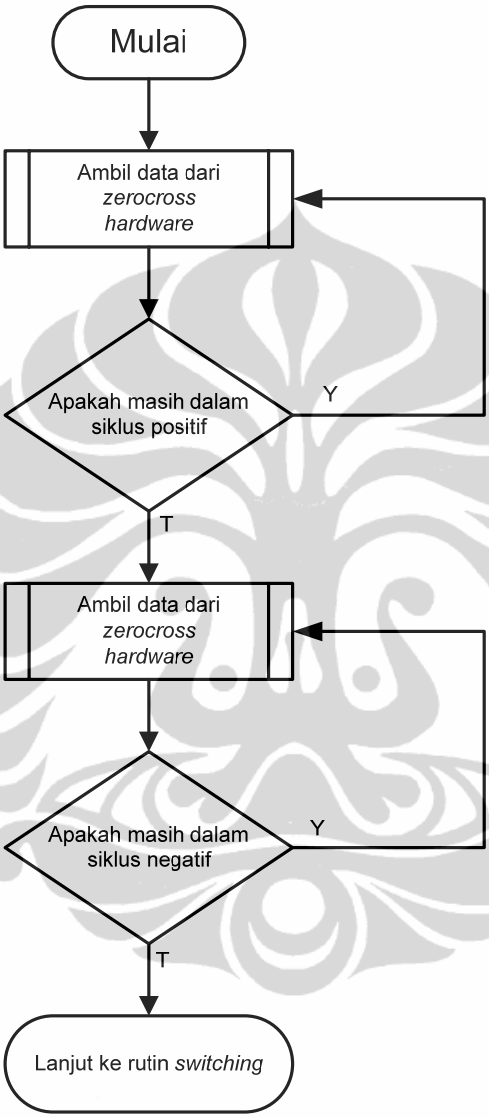
1. Sinus peralihan dari siklus positif ke negatif (fasa 180˚).
2. Sinus peralihan dari siklus negatif ke positif (fasa 0˚ atau 360˚).

Yang dipakai untuk mendeteksi fasa nol adalah yang ke dua. Rangkaian elektronik *zero cross* terlihat seperti pada Gambar 3.6. Konsepnya adalah membandingkan sinyal sinus dengan *ground* sinyal tersebut. Sinyal sinus di hubungkan dengan pin *non-inverting*, sedangkan ground sinyal di hubungkan dengan pin *inverting*. Output *comparator* di hubungkan dengan portC.0 Atmega16.



Gambar 3.6 *Hardware* rangkaian deteksi *zero cross*

Untuk menselaraskan kerja *hardware* ini maka harus diimbangi dengan *software*, oleh sebab itu dirancang suatu algoritma yang mampu mengenali fasa positif dan fasa negatif sinyal.



Gambar 3.7 *Flowchart* deteksi *zero cross*

Deskripsi algoritma yang akan dirancang adalah sebagai berikut:

1. Mulai.
2. Ambil data dari *zero cross* hardware.
3. Apakah masih dalam siklus positif. Jika sinus yang terdeteksi masih dalam siklus positif maka kembali ke langkah 2. Jika sinus yang terdeteksi dalam siklus negatif maka lanjut ke langkah 4.

4. Ambil data dari *zero cross hardware*.
5. Apakah masih dalam siklus negatif. Jika sinus yang terdeteksi masih dalam siklus negatif maka kembali ke langkah 4. Jika sinus yang terdeteksi dalam siklus positif maka lanjut ke langkah 6.
6. Lanjut ke rutin *switching*.
7. Selesai.

Deskripsi algoritma diatas bila diimplementasikan dalam *flowchart* (diagram alir) akan terlihat seperti Gambar 3.7.

Setelah deskripsi algoritma dan *flowchart* dibuat, maka langkah selanjutnya yang dilakukan adalah implementasi ke dalam bahasa pemrograman. Pada mulanya penelitian ini dibuat dalam bahasa C, namun karena lambat maka rutin *zero cross* dibuat dalam bahasa *assembler*. Hal ini dilakukan karena bahasa *assembler* menggunakan waktu eksekusi yang jauh lebih cepat.

Algoritma dan *flowchart* diatas bila di implementasikan dalam bahasa *assembler* akan terlihat seperti di bawah ini.

```
#asm
push r16        ; pindahkan r16 ke stacktest_cepat:
in r16,PinC     ; ambil masukan dari portC.0 , pindahkan ke r16
sbrc r16,0      ; skip if bit  r16,0  is clear
jmp test_cepat  ; jika  masih high maka lompat ke lagi_cepat
lagi_cepat:
in r16,PinC     ; ambil masukan darai portC.0 , pindahkan ke r16
sbrs r16,0      ; skip if bit  r16,0  is clear
jmp lagi_cepat  ; jika  masih high maka lompat ke  lagi_cepat
pop r16         ; pindahkan stack ke r16
#endasm
```

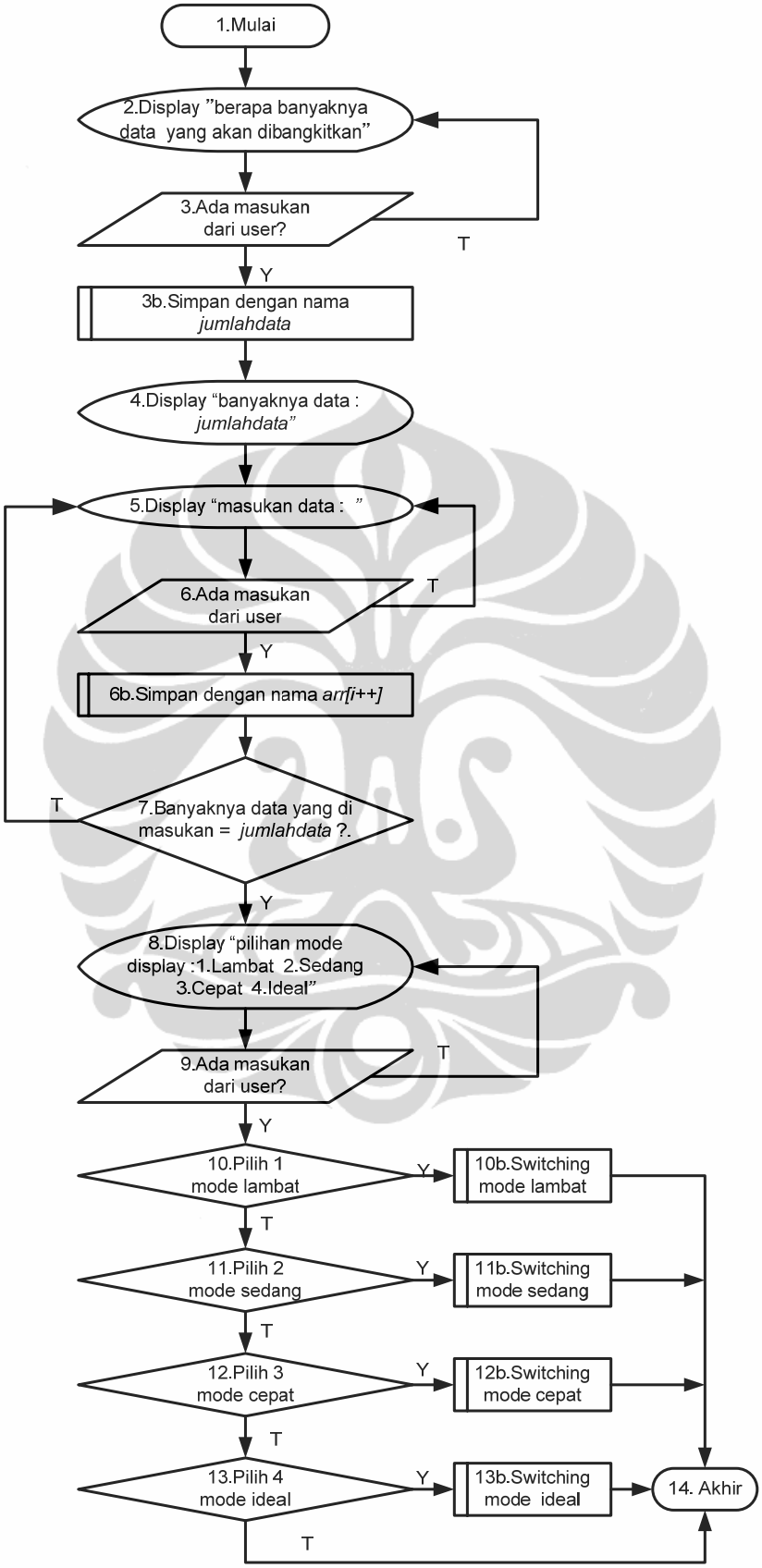
Deskripsi rutin *zero cross* adalah sebagai berikut:

1. Dorong data r16 ke *stack pointer* (*stack pointer* akan digunakan sementara untuk operasi pendeteksian *zero cross* pada akhir program akan dikembalikan seperti semula).
2. Ambil masukan dari portC.0 kemudian pindahkan ke r16.

3. Chek apakah bit r16,0 *low*. Jika *low* maka loncati perintah dibawahnya (menuju langkah ke empat). Namun jika *high* maka lanjutkan perintah di bawahnya (kembali ke langkah ke dua). Langkah ini merupakan pengecekan apakah sinus berada dalam siklus positif. Logika *low* menunjukkan bahwa sinyal sinus yang terdeteksi berada dalam siklus negatif. Logika *high* menunjukkan bahwa sinyal sinus yang terdeteksi berada dalam siklus positif. Dengan kata lain : Chek apakah bit r16,0 berada dalam siklus negatif. Jika negatif maka loncati perintah dibawahnya (menuju langkah ke empat). Namun jika berada dalam silkus positif maka lanjutkan perintah di bawahnya (kembali ke langkah ke dua).
4. Ambil masukan dari portC.0 kemudian pindahkan ke r16.
5. Chek apakah bit r16,0 *high*. Jika *high* maka loncati perintah dibawahnya (akhiri rutin *zero cross* lanjutkan ke rutin pensaklaran atau langkah 6). Namun jika *high* maka lanjutkan perintah di bawahnya (kembali ke langkah ke 4). Langkah ini merupakan pengecekan apakah sinus berada dalam siklus negatif. Logika *low* menunjukkan bahwa sinyal sinus yang terdeteksi berada dalam siklus negatif. Logika *high* menunjukkan bahwa sinyal sinus yang terdeteksi berada dalam siklus positif.
6. Tarik (ambil data dari) dari *stack pointer* kembalikan r16 (r16 telah selesai digunakan operasi maka dikembalikan seperti semula).

3.2.3. Perancangan Generator 3-bit *programmable*

Generator 3-bit merupakan rutin program yang membangkitkan data 3 bit (000, 001, 010, 011, 100, 101, 110, 111) sesuai masukan yang di berikan. Generator 3-bit merupakan sumber modulasi generator *8–PSK* untuk selanjutnya dimodulasi dengan menggunakan sinyal sinus. Generator 3-bit ini akan membangkitkan data *biner* 3-bit secara periodik sesuai masukannya.



Gambar 3.8 *Flowchart* generator 3-bit *programmable*

Deskripsi algoritma yang dirancang adalah sebagai berikut:

1. Mulai.
2. Tampilkan ke *LCD* "berapa banyaknya data yang akan dibangkitkan".
3. Meminta masukan dari pemakai berapa banyaknya data yang akan dibangkitkan oleh generator 3-bit. Jika belum ada masukan dari pemakai, maka kembali ke langkah 2. Jika pemakai telah memberikan masukan, maka simpan dengan nama *jumlahdata,* lanjut langkah 4.
4. Tampilkan ke *LCD* "banyaknya data : *jumlahdata"* (sesuai *input* dari langkah 3).
5. Tampilkan ke *LCD* "masukan data : "
6. Meminta masukan dari pemakai. Jika belum ada masukan dari pemakai, maka kembali ke langkah 5. Jika pemakai telah memberikan masukan, maka simpan dengan nama *arr[i++],* lanjut langkah 7.
7. Chek apakah banyaknya data yang di masukan telah sama dengan *jumlahdata* ?. Jika belum sama maka kembali ke langkah 5, jika sudah sama maka lanjut ke langkah 8.
8. Tampilkan ke *LCD* "Tentukan pilihan mode display modulasi *8-PSK* (1.Lambat 2.Sedang 3.Cepat 4.Ideal)". Ini adalah *option* dari mikrokontroler bagi pemakai untuk menetukan mode display *8-PSK* yang akan dipakai.
9. Meminta masukan dari pemakai. Jika belum ada masukan dari pemakai, maka kembali ke langkah 8. Jika pemakai telah memberikan masukan, maka lanjut langkah 10 .
10. Jika pilih 1 maka jalankan rutin pensaklaran mode lambat. Mode lambat merupakan mode untuk menampilkan modulasi *8 Gray-PSK* dimana 3-bit kode *Gray* di wakili oleh 200 gelombang sinus, hal ini bertujuan untuk mudah di amati dengan cermat melalui mata langsung.
11. Jika pilih 2 maka jalankan rutin pensaklaran mode sedang. Mode sedang merupakan mode untuk menampilkan modulasi *8 Gray-PSK* dimana 3-bit kode *Gray* di wakili oleh 3 gelombang sinus.

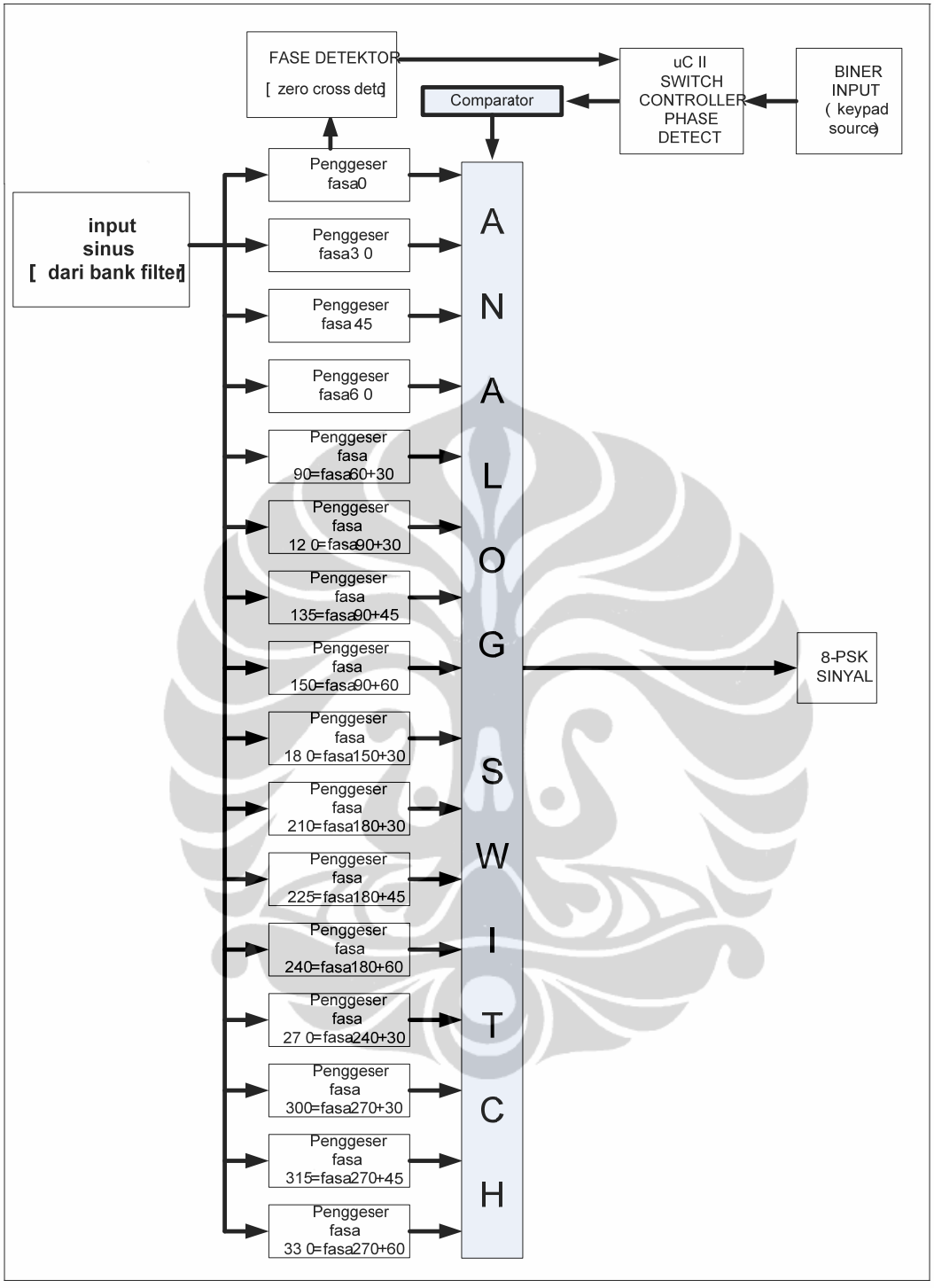
12. Jika pilih 3 maka jalankan rutin pensaklaran mode cepat. Mode cepat merupakan mode untuk menampilkan modulasi *8 Gray-PSK* dimana 3-bit kode *Gray* di wakili oleh 2 gelombang sinus.
13. Jika pilih 4 maka jalankan rutin pensaklaran mode ideal. Mode ideal merupakan mode untuk menampilkan modulasi *8 Gray-PSK* dimana 3-bit kode *Gray* di wakili oleh 1 gelombang sinus, hal ini bertujuan untuk memperlihatkan kondisi ideal pemodulasian *8 Gray-PSK*, selain itu untuk menghemat waktu dalam transmisi data.

Deskripsi algoritma Generator 3-bit diatas, bila diimplementasikan dalam *flowchart* (diagram alir) akan terlihat seperti Gambar 3.8. Setelah deskripsi algoritma dan *flowchart* dibuat, maka langkah selanjutnya yang dilakukan adalah implementasi ke dalam bahasa pemrograman. Algoritma dan *flowchart* generator 3-bit diatas, bila diimplementasikan dalam bahasa pemrograman akan terlihat pada lampiran.

3.2.4 Pensaklaran

Setelah perancangan *zero cross* dan generator 3-bit dibuat, maka langkah selanjutnya adalah melakukan pensaklaran sinyal yang telah digeser, sesuai dengan *input* yang diberikan. Blok diagram pen-saklar-an nya adalah seperti Gambar 3.9 di bawah ini. File **switch1.SCH** merupakan rangkaian pensaklaran (*switching*) yang berfungsi sebagai rangkaian yang akan men-*switch* fasa mana saja yang *ON* yang bersesuaian dengan informasi yang akan dimasukan.

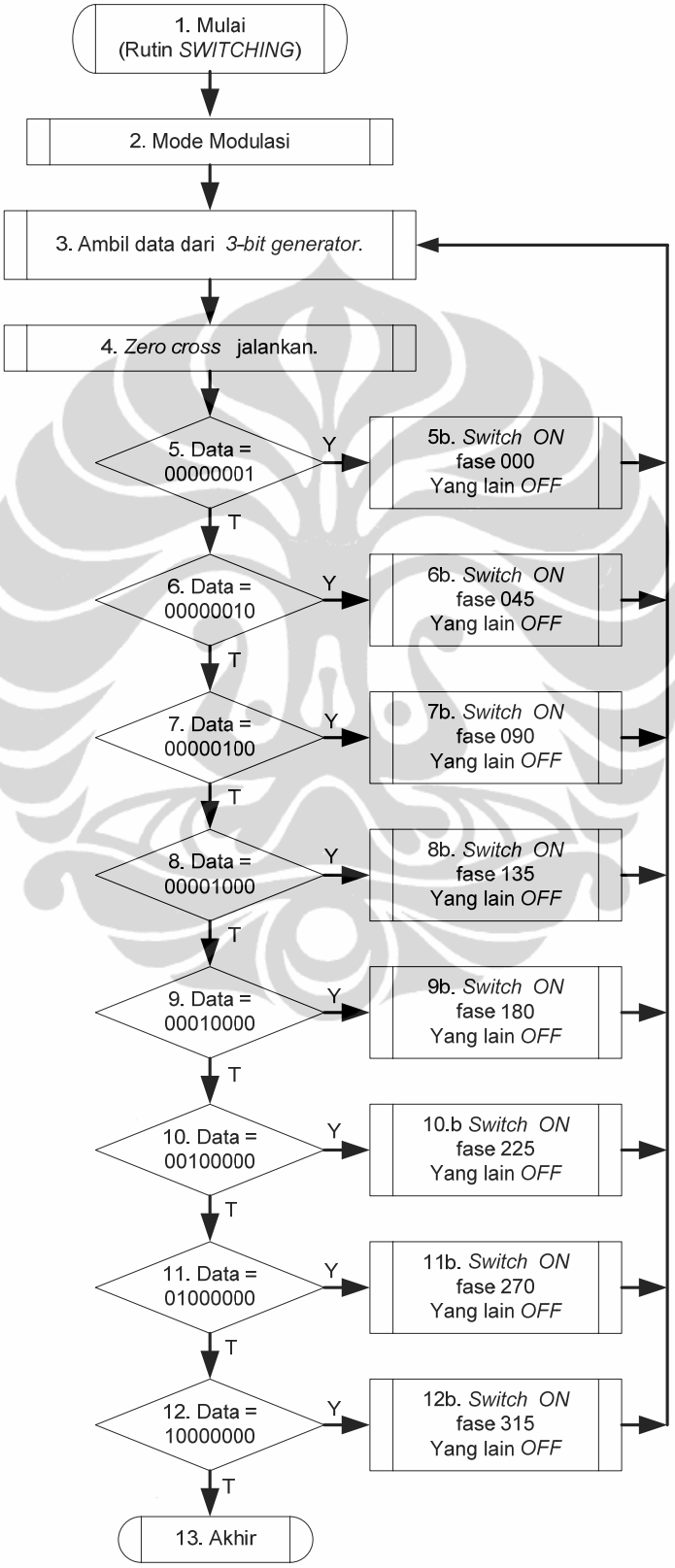
Rangkaian ini di kontrol melalui mikrokontroler Atmega16 port D. Dasar rangkaian ini adalah IC *analog switch*. *IC analog switch* memerlukan logika *CMOS* (+5V dan -5V) untuk men-*drive switch* pengontrolnya, oleh karena itu tegangan *TTL* dari mikrokontroler harus melalui rangkaian *comparator* yang akan mengubah logika *TTL* (0V/ *LOW* dan 5V/ *HIGH*) menjadi logika *CMOS* (-5 V / *LOW* dan +5 V / *HIGH*) agar dapat men-*drive IC analog switch*. Jika kedua hal diatas telah dibuat simulasi dan skematiknya maka selanjutnya adalah perancangan *hardware* yang didasarkan atas skematik.



Gambar 3.9 Blok Pensaklaran sinyal untuk menghasilkan *8-Phase Shift Keying*

Cara kerja rangkaian tersebut adalah: pertama pengguna memasukan data *biner* 3 bit misal saja 110, 100, 010, setelah di masukan maka mikrokontroler akan mendeteksi sinyal kapan sinus fasa nol menyentuh sumbu X (dengan

bantuan rangkaian *zero cross*) jika mikrokontroler telah mengetahui bahwa fasa 0˚ telah melewati sumbu 0 maka selanjutnya mikrokontroler akan men-*switch* fasa-fasa yang bersesuaian dengan *input biner* yang dimasukan.



Gambar 3.10 *Flowchart* Rutin Pensaklaran fasa.

Algoritma yang di rancang pada rutin pensaklaran adalah sebagai berikut:

1. Mulai.
2. Tampilkan mode modulasi.
3. Ambil data dari *3-bit generator*.
4. *Zero Cross* jalankan.
5. *Switch* fasa mana yang bersesuaian dengan data 3 bit.
6. Kembali ke langkah 3 (*looping*).
7. Selesai.

Hal yang tidak kalah penting adalah pembuatan *software* pen-saklaran. Pada tahapan ini hal yang dilakukan adalah perancangan *flowchart* dan algoritma. Algoritma pensaklaran bila di buat *flowchart* –nya terlihat pada Gambar 3.10. Rutin pensaklaran-nya adalah seperti di bawah ini (untuk mode 200 sinus mewakili 3 bit):

```
void cetakarray_ideal(int *arr,int n)
     {
      int i;
      lcd_gotoxy(0,0);
      lcd_putsf("zerocross triggr");
      lcd_gotoxy(0,1);
      lcd_putsf("Lihat Osciloskop");delay_ms(250);

       for (i=0; i< n;i++)
            {    baca_diregister_nya=arr[i];
///################for 10 siklus+ getwrn + 019

//#################################################################
//di BAWAH sini ZERO CROSS DETECTOR harus men-trigger
//bahasa asemblrer
        #asm
        push r16
                testlagi_ideal:
                in r16,PinC
                        ;sbic portC,0
                sbrc r16,0
                        jmp testlagi_ideal
                lagi_ideal:
                in r16,PinC
                        ;sbis portC,0
                sbrs r16,0
                        jmp lagi_ideal
        ;pop r16
        #endasm
//di ATAS  sini ZERO CROSS DETECTOR harus men-trigger

//#################################################################
```

```
        #asm
        sbrc  r6,0
                jmp fasa_000
        sbrc  r6,1
                jmp fasa_045
        sbrc  r6,2
                jmp fasa_090
        sbrc  r6,3
                jmp fasa_135
        sbrc  r6,4
                jmp fasa_180
        sbrc  r6,5
                jmp fasa_225
        sbrc  r6,6
                jmp fasa_270
        sbrc  r6,7
                jmp fasa_315

        jmp akhiri_rutin

;????????????????????????????????????????????????
;????????????????????????????????????????????????
        fasa_000:
                ldi r16,1
                out portD,r16
                jmp akhiri_rutin
        fasa_045:
                ldi r16,2
                out portD,r16
                jmp akhiri_rutin
        fasa_090:
                ldi r16,4
                out portD,r16
                jmp akhiri_rutin
        fasa_135:
                ldi r16,8
                out portD,r16
                jmp akhiri_rutin
        fasa_180:
                ldi r16,16
                out portD,r16
                jmp akhiri_rutin
        fasa_225:
                ldi r16,32
                out portD,r16
                jmp akhiri_rutin
        fasa_270:
                ldi r16,64
                out portD,r16
                jmp akhiri_rutin
        fasa_315:
                ldi r16,128
                out portD,r16
                jmp akhiri_rutin

        akhiri_rutin:
        ;nop
        pop r16
        #endasm
```

```
       // ;k++;
        //;if (k==2)  {

if(i==(n-1)) {i=(i-n);}
        //  k=0;}

           }
     }
```

Ada 4 ide pokok diskripsi tentang rutin *software* diatas:

1. Rutin *zero cross* yang berfungsi memberikan info bagi mikrokontroler bahwa sinyal telah menyentuh tegangan nol.
2. Pembangkitkan data yang 3 bit yang telah disimpan sebelumnya dan dilanjutkan men-switch fasa yang bersesuaian.
3. Pensaklaran fasa sesuai dengan data bit yang dibangkitkan generator 3 bit.
4. *Looping* agar data 3 bit yang dibangkitkan periodik.

Konklusi Perancangan

Dalam penelitian ini terdapat hal-hal yang perlu dipahami agar tidak ada kekacauan antara konseptual (prinsipil) dan teknikal:

1. Pada perancangan rutin *software zero cross* perlu di perhatikan bahwa register yang di pakai untuk pendeteksian bit *zero cross* adalah bebas tidak harus r16. Hal yang perlu dicermati adalah sebelum pemakaian register tersebut harus ada *PUSH* dan sesudah pemakaian harus di akhiri dengan *POP*. Hal ini bertujuan mengembalikan nilai register tersebut ke keadaan semula.
2. Pada rangkaian *analog switch* menggunakan *comparator*. Resistor yang digunakan pada pin *inverting comparator* pada rangkaian *analog switch* haruslah di atur sedemikian rupa sehingga menghasilkan pembagi tegangan sebesar 2,7 V terhadap *ground*, dan dalam perhitungannya perlu memperhatikan nilai *Vdd* dan *Vss* yang dipakai. Hal ini semata-mata untuk memberi batas *threshold* yang tegas antara logika *TLL low* (-5V) dan *high* (+5V).