

BAB 3

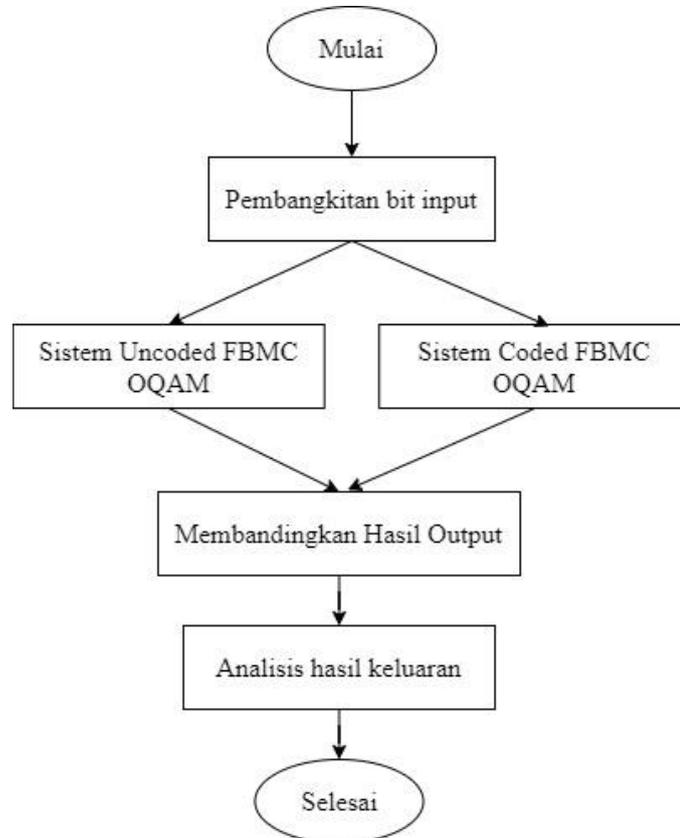
METODOLOGI PENELITIAN

3.1 Alat Yang Digunakan

Penelitian ini bersifat simulasi untuk melihat unjuk kerja penggunaan pengkodean kanal kode konvolusi pada sistem FBMC menggunakan modulasi OQAM. Alat yang digunakan untuk penelitian ini yaitu *software* Matlab R2017b.

3.2 Alur Penelitian

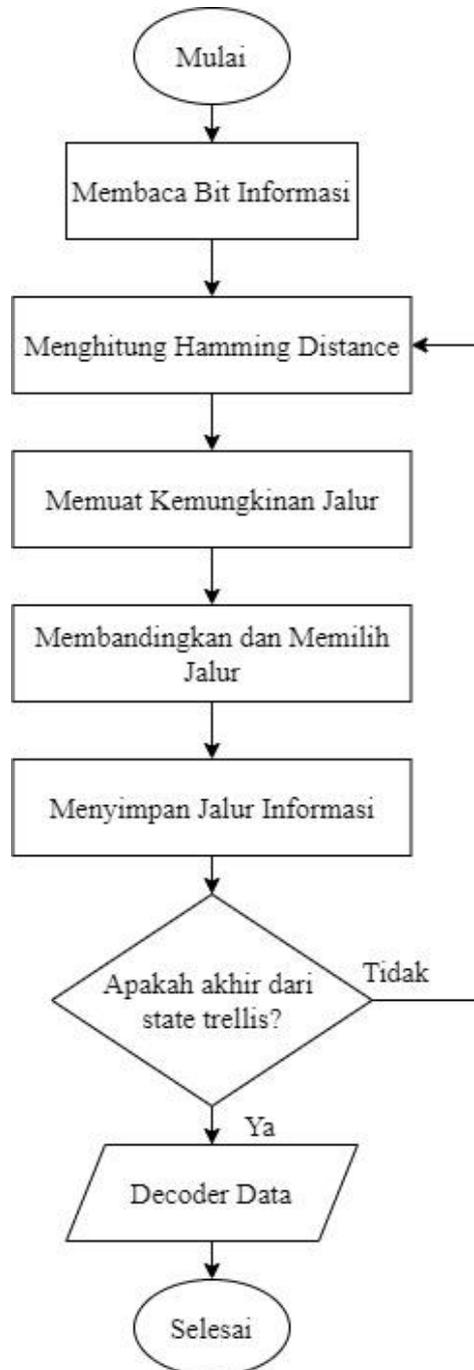
Rancangan proses pengerjaan pada penelitian ini mengenai simulasi sistem FBMC OQAM baik dengan pengkodean kanal menggunakan kode konvolusi maupun tidak menggunakan pengkodean kanal mengacu pada diagram alir yang ditunjukkan pada Gambar 3.1. Sedangkan pada Gambar 3.2 merupakan proses dari *decoder* algoritma viterbi.



Gambar 3.1 Alur penelitian

Pada penelitian yang dilakukan, pertama adalah pembangkitan bit *input*. Pembangkitan bit *input* sebanyak 45000 bit digunakan sebagai masukan pada

sistem *uncoded* FBMC OQAM dan sistem *coded* FBMC OQAM dengan kode konvolusi. *Output* yang dihasilkan oleh kedua sistem tersebut, akan dibandingkan berdasarkan dengan parameter SNR terhadap BER dan kapasitas kanal. Selanjutnya akan dilakukan analisis guna mengetahui manakah yang lebih baik digunakan.



Gambar 3. 2 Alur Decoder Algoritma Viterbi

Proses *decoder* dari algoritma viterbi yang pertama adalah proses pembacaan bit informasi. Pembacaan bit informasi dilakukan sesuai dengan *code*

rate yang digunakan yang kemudian dilakukan perhitungan *hamming distance*. *Hamming distance* merupakan proses pencocokan bit informasi dengan ketentuan bit *decoder* serta mencari bit dengan kesalahan yang paling sedikit. Hasil dari *hamming distance* akan membuat kemungkinan jalur bit. Setelah terbentuk beberapa kemungkinan jalur bit, selanjutnya akan dibandingkan serta dipilih jalur terbaik yang akan digunakan. Jalur yang telah terpilih akan disimpan dalam jalur informasi dan dapat menghasilkan *state* untuk bit informasi selanjutnya. Apabila masih terdapat bit informasi yang belum dikodekan kembali maka proses akan dimulai lagi dari menghitung *hamming distance*. Jika seluruh bit informasi sudah dikodekan kembali maka akan menghasilkan bit informasi yang telah dikodekan.

3.3 Parameter Simulasi

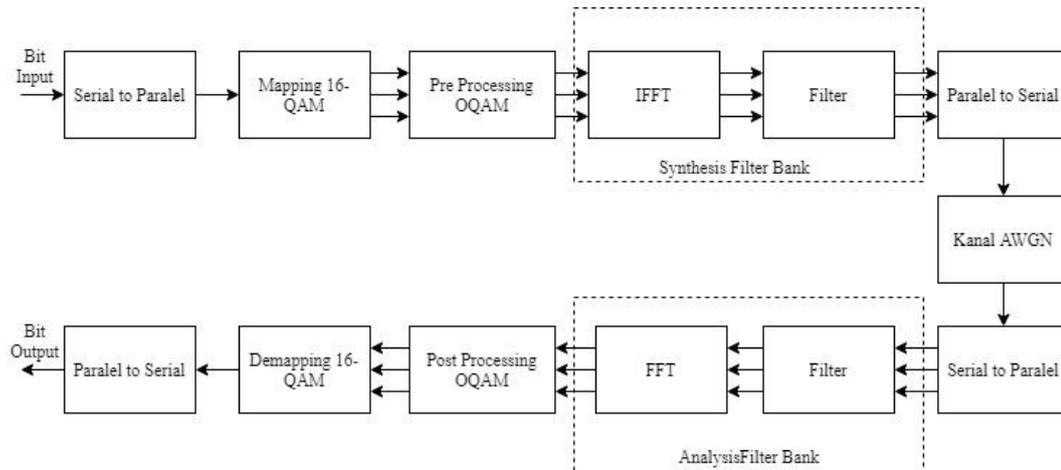
Parameter simulasi yang digunakan pada penelitian ini sebagai berikut :

Tabel 3.1 Parameter simulasi

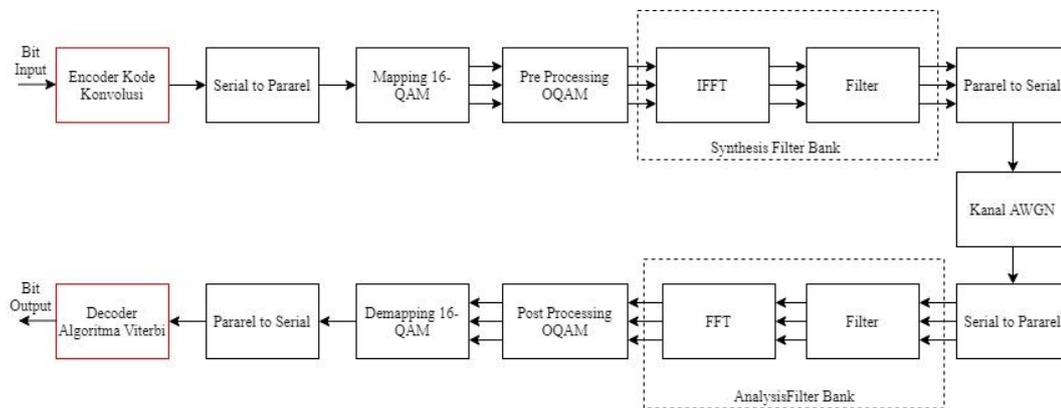
Parameter	Nilai
<i>Bit Input</i>	45000 bit
<i>Code Rate</i> Konvolusi	1/2
<i>Generator Polinomial</i>	[7 5]
Algoritma Viterbi	<i>Hard Decision</i>
<i>Mapping</i>	16 QAM
Jumlah Antena Pengirim	1
Jumlah Antena Penerima	1

3.4 Pemodelan Sistem

Berikut pemodelan sistem yang digunakan pada penelitian ini :



Gambar 3.3 Pemodelan sistem *uncoded* FBMC OQAM



Gambar 3.4 Pemodelan sistem *coded* FBMC OQAM

Pemodelan Gambar 3.4 merupakan pemodelan seperti pada Gambar 3.3, namun terdapat perbedaan pada sisi pengirim dan penerima. Perbedaan tersebut yaitu terdapat penambahan blok *encoder* pada sisi pengirim dan blok *decoder* pada sisi penerima. Berikut ini penjelasan dari kedua blok diagram tersebut:

3.4.1 Pengirim

3.4.1.1. Data Masukan

Data masukan pada penelitian ini berupa simbol acak 0 dan 1 yang merupakan simbol dari sinyal digital. Bit data yang dibangkitkan sebanyak 45000 bit. Berikut program yang digunakan untuk membangkitkan bit biner :

```
data_input=randi([0 1],1,45000);
```

Dimana *data_input* merupakan hasil dari pembangkitan bit secara acak menggunakan fungsi *randi* dengan angka yang dibangkitkan bernilai 0 dan 1 serta berukuran 1x45000.

3.4.1.2. Kode Konvolusi

Code rate konvolusi yang digunakan untuk penelitian ini yaitu $\frac{1}{2}$ dengan panjang memori 3 dan *generator polinomial* yang digunakan [7 5]. Kode program yang digunakan untuk mengkodekan data masukan sebagai berikut:

```
t=poly2trellis(3,[7 5]);
codeword = convenc(data_input,t);
```

Dimana *poly2trellis* digunakan untuk membuat *trellis* dengan panjang memori 3 dan generator polinomial [7 5] serta disimpan dalam variabel *t*. *Codeword* merupakan hasil dari pengkodean kode konvolusi dengan *data_input* sebagai data inputnya sesuai dengan *trellis t*. Berikut tabel konvolusi yang dihasilkan dari *encoder* kode konvolusi dengan *code rate* $\frac{1}{2}$ dengan generator polinomial [7 5]:

Tabel 3.2 Tabel konvolusi dengan *code rate* $\frac{1}{2}$.

Keadaan saat ini		Input	Output			Keadaan berikutnya	
0	0	0	0	0	0	0	
		1	1	1	1	0	
0	1	0	1	1	0	0	
		1	0	0	1	0	
1	0	0	1	0	0	1	
		1	0	1	1	1	
1	1	0	0	1	0	1	
		1	1	0	1	1	

3.4.1.3. Serial to Paralel

Data yang telah dikodekan menggunakan kode konvolusi masih dalam bentuk serial, untuk dapat diproses selanjutnya maka data dikonversi dari serial menjadi paralel dengan fungsi berikut :

```
sp_tx = reshape(codeword, ml, length(codeword)/ml);
```

Fungsi *reshape* digunakan untuk mengubah ukuran matrik *codeword* menjadi matrik *sp_tx* dengan ukuran kolom *ml* dan baris *codeword/ml*.

3.4.1.4. Mapping 16-QAM

Pada proses ini *input* yang berupa bilangan *biner* dikonversi menjadi bilangan kompleks yang terdiri dari bagian *inphase* dan bagian *quadrature*. Proses *mapping* yang digunakan pada penelitian ini menggunakan *mapping* 16-QAM, dimana setiap 4 bit bilangan biner akan menghasilkan 1 bilangan kompleks. Berikut kode program yang digunakan:

```
for c=1:(length(sp_tx))
    qammod(c, :)=(1/sqrt(10))*((1-2*(sp_tx(c,4)))*(2-(1-2*(sp_tx(c,2))))+sqrt(-1)*(1-2*(sp_tx(c,3)))*(2-(1-2*(sp_tx(c,1)))));
end
```

Tabel 3.3 Mapping 16 QAM

Bit Biner				Bilangan Kompleks
Bit 4	Bit 3	Bit 2	Bit 1	
0	0	0	0	0,3162+0,3162i
0	0	0	1	-0,3162+0,3162i
0	0	1	0	0,3162-0,3162i
0	0	1	1	-0,3162-0,3162i
0	1	0	0	0,9486+0,3162i
0	1	0	1	-0,9486+0,3162i
0	1	1	0	0,9486-0,3162i
0	1	1	1	-0,9486-0,3162i
1	0	0	0	0,3162+0,9486i
1	0	0	1	-0,3162+0,9486i
1	0	1	0	0,3162-0,9486i
1	0	1	1	-0,3162-0,9486i
1	1	0	0	0,9486+0,9486i
1	1	0	1	-0,9486+0,9486i
1	1	1	0	0,9486-0,9486i
1	1	1	1	-0,9486-0,9486i

Tabel 3.3 merupakan ketentuan konversi dari sinyal yang masih dalam bentuk bit biner menjadi bilangan kompleks. Hasil dari konversi didapat menggunakan Persamaan 2.3 dan menghasilkan 16 kemungkinan konversi sinyal.

3.4.1.5. Pra Pengolahan OQAM

Pada proses pra pengolahan OQAM, satu bilangan kompleks dipisahkan menjadi 2 simbol baru dimana satu simbol terdiri dari bagian riil dan yang satunya terdiri dari bagian imajiner. Berikut kode program dari pra pengolahan OQAM :

```
for n = 1 : length(qammod_atas)
    if (rem(n, 2) == 1) %odd (ganjil)
        oqam_pre(n,1)=imag(qammod_atas(n))*((j).^n);
        oqam_pre(n,2)=real(qammod_atas(n))*((j).^n);
    else %even (genap)
        oqam_pre(n,1)=real(qammod_atas(n))*((j).^n);
        oqam_pre(n,2)=imag(qammod_atas(n))*((j).^n);
    end
end
```

Pada subkanal ganjil simbol kompleks akan diambil bagian imajiner terlebih dahulu kemudian dikalikan dengan nilai (j) untuk mendapatkan simbol kompleks baru yang pertama. Sedangkan pada subkanal genap, simbol kompleks diambil bagian riil terlebih dahulu dan dikalikan dengan nilai (j) untuk mendapatkan simbol kompleks yang pertama. Simbol baru yang kedua akan diletakan diakhir urutan simbol baru yang pertama. Pada simbol baru yang kedua akan diambil bagian riil pada subkanal ganjil dan bagian imajiner pada subkanal genap. Kemudian bagian tersebut dikalikan dengan nilai (j) untuk mendapatkan simbol kompleks baru.

3.4.1.6. *Synthesis Filter Bank*

Proses *synthesis filter bank* terdiri dari proses IFFT (*Inverse Fast Fourier Transform*) dan proses filter. Pada proses IFFT memisahkan sinyal berdasarkan frekuensinya atau mengubah sinyal dari domain frekuensi ke domain waktu. Setelah sinyal dipisahkan berdasarkan frekuensinya kemudian sinyal difilter menggunakan filter ideal, dimana filter ideal memiliki karakteristik 1.

```
ifft_out = ifft(oqam_pre);
%PPN Filter
```

```

z=1; %contoh filter = 1
for x=1:length(ifft_out)

    ppn_sfb(x,:)=ifft_out(x)*z;
end

```

3.4.1.7. Paralel to Serial

Setelah data melalui proses *synthesis filter bank*, kemudian data yang dalam bentuk paralel diubah kedalam bentuk serial untuk ditransmisikan. Perubahan data dari paralel ke serial menggunakan fungsi berikut :

```
ps_tx = reshape(ppn_sfb,1,[]);
```

Dimana *ps_tx* merupakan hasil dari konversi data *ppn_sfb* paralel menjadi serial dengan ukuran 1 baris dan kolom tak hingga.

3.4.1.8. Kanal AWGN

Data yang ditransmisikan dari pengirim ke penerima melalui kanal, kanal yang digunakan pada penelitian ini yaitu kanal AWGN. Fungsi yang digunakan pada *software* matlab sebagai berikut :

```
rx = (1.*source1 + noise);
```

Dimana 1 merupakan karakteristik kanal AWGN dimana memiliki nilai 1, *source1* merupakan sinyal yang akan dikirimkan dan *noise* merupakan sinyal acak yang panjangnya telah disesuaikan dengan panjang *source1*.

3.4.2 Penerima

3.4.2.1. Serial to Paralel

Data yang diterima dari pengirim masih dalam bentuk serial, kemudian data diubah menjadi paralel menggunakan fungsi :

```
sp_rx = reshape(rx,[],1);
```

Dimana *sp_rx* merupakan hasil konversi dari serial ke paralel sinyal atau data *rx* dengan baris tak hingga dan 1 kolom.

3.4.2.2. Analysis Filter Bank

Sinyal yang berbentuk paralel kemudian difilter menggunakan filter ideal (filter yang sama dengan proses *synthesis filter bank*). Kemudian sinyal yang masih terpisah berdasarkan frekuensinya digabungkan kembali menggunakan FFT (mengubah sinyal dari domain waktu ke domain frekuensi).

```
z=1; %contoh filter = 1
for x=1:length(sp_rx)
    ppn_afb(:,x)=sp_rx(x)*z;
end
%Transformation Block
fft_out = fft(ppn_afb);
```

3.4.2.3. Pasca Pengolahan OQAM

Proses pasca pengolahan OQAM data yang dalam bentuk riil diubah menjadi kompleks. Satu simbol kompleks membutuhkan 2 simbol riil yang saling berurutan dan salah satu simbol riilnya akan dikalikan dengan j (imajiner). Berikut program untuk pasca pengolahan OQAM:

```
for n = 1 : length(ppn_afb_out)
    if (rem(n, 2) == 1) %odd (ganjil)
        oqam_post=ppn_afb_out*((-1*j).^n);
        oqam_post=real(oqam_post);
        oqam_post_atas(n,1)=oqam_post(n,2)+j*oqam_post(n,
        1);
    else %even (genap)
        oqam_post=ppn_afb_out*((-1*j).^n);
        oqam_post=real(oqam_post);
        oqam_post_atas(n,1)=oqam_post(n,1)+j*oqam_post(n,
        2);
    end
end
end
```

Pada pasca pengolahan OQAM merupakan kebalikan dari pra pengolahan OQAM. Simbol kompleks akan dikalikan dengan *conjugate* dari $\theta_{k,n}$ untuk mendapatkan bagian riil. Kemudaaian bagian riil tersebut disusun untuk membentuk

simbol kompleks. Satu simbol kompleks membutuhkan 2 bagian riil yang berurutan, urutan dari bagaian riil sama seperti urutan pada pra pengolahan OQAM.

3.4.2.4. Demapping

Simbol kompleks yang didapatkan pada proses pasca pengolahan OQAM akan diubah kembali menjadi biner. Satu simbol kompleks (*inphase* dan *quadrature*) akan menghasilkan 4 bit biner. Proses *demapping* dilakukan dengan menghitung jarak sinyal dengan titik pada diagram konstelasi. Sinyal akan mengikuti titik yang berada paling dekat dengan sinyal tersebut. Proses ini merupakan kebalikan dari proses *mapping*. Berikut ketentuan konversi dari bilangan kompleks bagian riil dan imajiner menjadi bentuk bit biner pada tabel *demapping* dari 16 QAM:

Tabel 3. 4 Demapping 16 QAM

Riil	Imajiner	Output
$0 < x < 0,6324$	$0 < y < 0,6324$	0000
$-0,6324 < x < 0$	$0 < y < 0,6324$	0001
$0 < x < 0,6324$	$-0,6324 < x < 0$	0010
$-0,6324 < x < 0$	$-0,6324 < x < 0$	0011
$x > 0,6324$	$0 < y < 0,6324$	0100
$x < -0,6324$	$0 < y < 0,6324$	0101
$x > 0,6324$	$-0,6324 < x < 0$	0110
$x < -0,6324$	$-0,6324 < x < 0$	0111
$0 < x < 0,6324$	$y > 0,6324$	1000
$-0,6324 < x < 0$	$y > 0,6324$	1001
$0 < x < 0,6324$	$y < -0,6324$	1010
$-0,6324 < x < 0$	$y < -0,6324$	1011
$x > 0,6324$	$y > 0,6324$	1100
$x < -0,6324$	$y > 0,6324$	1101
$x > 0,6324$	$y < -0,6324$	1110
$x < -0,6324$	$y < -0,6324$	1111

3.4.2.5. Algoritma Viterbi

Algoritma viterbi berfungsi untuk mengembalikan data yang ditelah dikodekan dan tercampur dengan *noise*. Algoritma viterbi menyesuaikan dengan *code rate* dan generator polinomial dari kode konvolusi yang digunakan pada proses pengiriman. *Code rate* $\frac{1}{2}$ maka setiap 2 bit yang diterima akan menghasilkan 1 bit *output* informasi. Berikut kode program yang digunakan:

```
pesan_terima=vitdec(data_output,t,tb,'trunc','hard');
```

Dimana *pesan_terima* merupakan hasil dari *decoder* dengan algoritma viterbi dari *data_output* dengan menggunakan *trellis* *t*, panjang penulusuran *tb*, mode operasi *trunc* dan jenis *hard decision*

