BAB III

METODOLOGI DAN PERANCANGAN SISTEM

3.1 Metodologi Penelitian

Beberapa metode penelitian yang digunakan untuk melakukan penelitian ini, antara lain:

a. Studi Literatur

Dalam studi literatur, dilakukan pembelajaran terhadap konsep komunikasi pada UTAR NoC, bagaimana proses arbitrasi pada UTAR NoC bekerja, dan beberapa *tools* yang diperlukan untuk melakukan proses verifikasi. Beberapa algoritma yang digunakan untuk melakukan proses arbitrasi juga dipelajari, seperti age-based [7] dan fixed priority [5].

b. Perancangan Algoritma

Dalam tahapan ini, dilakukan perancangan algoritma *distance-based priority*. Proses arbitrasi dari algoritma *distance-based priority* akan dijelaskan dalam diagram alur, dari tahap masuknya *request* sampai dengan diberikan *grant*. Pemrograman menggunakan bahasa Verilog.

c. Implementasi dan Pengujian Sistem

Tahapan implementasi dan pengujian sistem dimulai dengan mengimplementasi algoritma yang telah dibuat ke dalam arbiter dari UTAR NoC, lalu melakukan verifikasi untuk membuktikan bahwa algoritma sudah berjalan dengan semestinya.

Pengujian dilakukan dengan menggunakan beberapa kasus dasar sampai dengan aplikasi-aplikasi dari MCSL NoC Traffic Pattern [12]. Hasil pengujian berupa jumlah clock yang diperlukan untuk menyelesaikan kasus, rata-rata *latency* dan *throughput* pada jaringan. *Latency* adalah waktu yang diperlukan agar sebuah paket sampai ke tujuan. *Throughput* adalah jumlah paket yang dapat terkirim dalam satuan waktu.

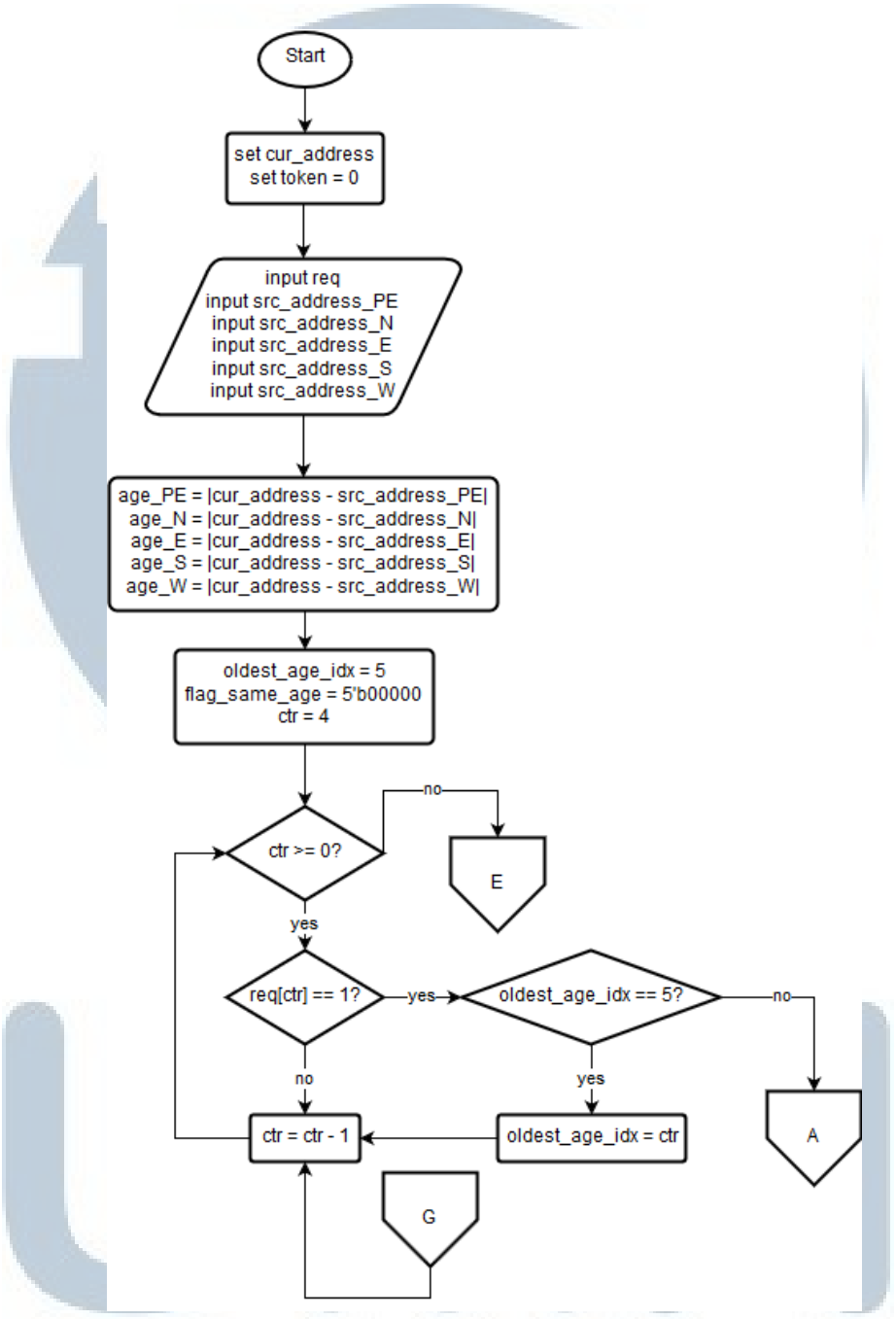
d. Analisis

Pada tahapan analisis dilakukan pengambilan hasil performa yaitu, *latency* dan *throughput* dari setiap aplikasi yang dijalankan. Hasil ini kemudian dibandingkan dengan hasil dari performa arbitrasi UTAR NoC yang menggunakan round robin.

3.2 Perancangan Algoritma

Algoritma distance-based dimulai dengan menghitung *age* (jumlah hop) dari paket. *Age* didapat dengan menghitung selisih koordinat dari *current router* dengan *source router*. Paket dengan nilai *age* paling tinggi akan diberikan *grant*. Jika terdapat dua atau lebih paket dengan nilai *age* tertinggi yang sama, *grant* akan ditentukan dengan algoritma round robin.

Proses awal yang dilakukan dalam algoritma *distance-based* digambarkan melalui diagram alir seperti pada Gambar 3.1.



Gambar 3.1 Diagram Alir Proses Awal

Pada Gambar 3.1, ketika ada *request*, arbiter akan menerima input berupa:

a. *req*, adalah *request* yang masuk ke arbiter, memiliki panjang 5 bit, dimana *req[4]* untuk *request* dari W, *req[3]* untuk *request* dari S, *req[2]* untuk *request* dari E, *req[1]* untuk *request* dari N, dan *req[0]* untuk *request* dari PE.

b. *src_address_PE*, adalah alamat router dari *request* PE.

c. *src_address_N*, adalah alamat router dari *request* N.

d. *src_address_E*, adalah alamat router dari *request* E.

e. *src_address_S*, adalah alamat router dari *request* S.

f. *src_address_W*, adalah alamat router dari *request* W.

Setelah menerima input, arbiter akan melakukan perhitungan berapa *age* (jumlah hop) dari paket. *Age* didapatkan dengan menghitung selisih dari koordinat router sekarang dengan koordinat router *source*.

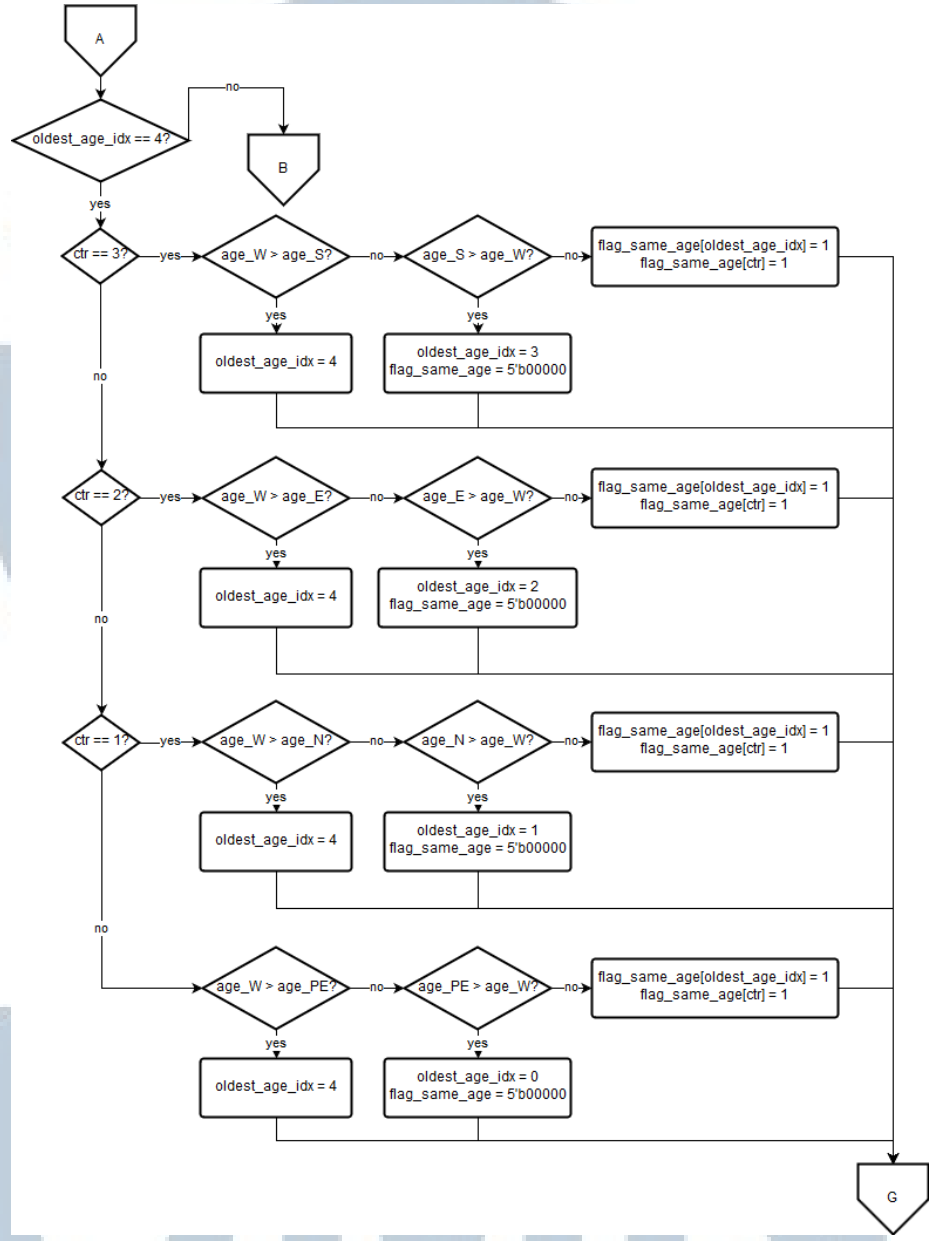
Proses selanjutnya adalah mengatur nilai awal beberapa variabel yang diperlukan, yaitu:

a. *oldest_age_idx*, merupakan index dari bit *req* yang memiliki *age* tertinggi.

b. *flag_same_age*, untuk menandakan bit mana saja jika terdapat dua atau lebih bit *req* yang memiliki *age* tertinggi yang sama.

c. *ctr*, sebagai *counter* untuk mengecek bit-bit *req*.

Arbiter mengecek bit per bit dari *req* sampai mendapatkan bit bernilai 1 dari kiri ke kanan. Index dari bit pertama yang ditemukan akan dijadikan sebagai *oldest_age_idx*. Untuk bit selanjutnya, akan dilakukan proses komparasi *age* yang digambarkan dalam diagram alir A (Gambar 3.2), B (Gambar 3.3), C (Gambar 3.4), dan D (Gambar 3.5).



Gambar 3.2 Diagram Alir A

Proses komparasi pada diagram alir A (Gambar 3.2) dimulai jika *oldest_age_idx* bernilai 4, dimana *age* tertinggi saat ini berada di W. Selanjutnya komparasi akan dilakukan dengan mengecek nilai *ctr*.

Jika *ctr* bernilai 3, maka *age_W* akan dibandingkan dengan *age_S*. Apabila *age_W* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_S* lebih

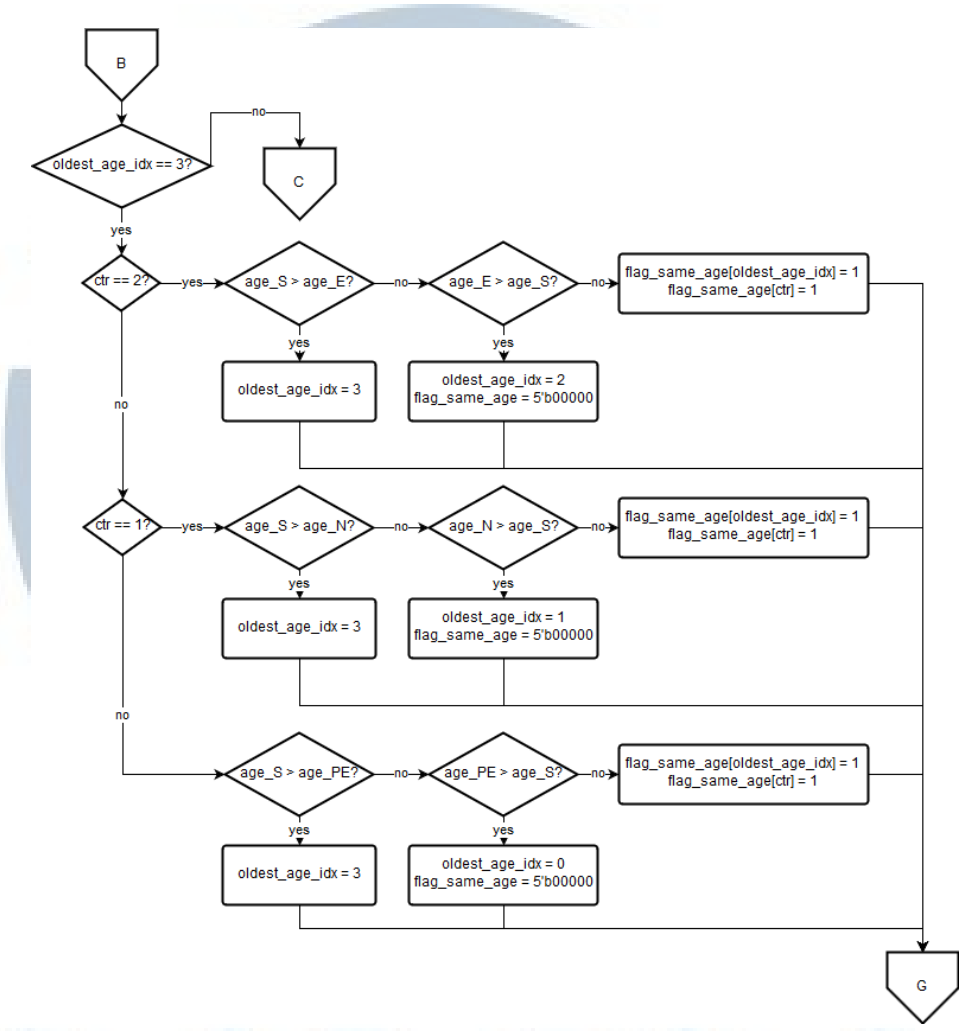
besar maka nilai *oldest_age_idx* menjadi 3 dan *flag_same_age* menjadi 5'b00000. Apabila *age_W* sama dengan *age_S* maka *flag_same_age* menjadi 5'b11000.

Jika ctr bernilai 2, maka *age_W* akan dibandingkan dengan *age_E*. Apabila *age_W* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_E* lebih besar maka nilai *oldest_age_idx* menjadi 2 dan *flag_same_age* menjadi 5'b00000. Apabila *age_W* sama dengan *age_E* maka *flag_same_age* menjadi 5'b10100.

Jika ctr bernilai 1, maka *age_W* akan dibandingkan dengan *age_N*. Apabila *age_W* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_N* lebih besar maka nilai *oldest_age_idx* menjadi 1 dan *flag_same_age* menjadi 5'b00000. Apabila *age_W* sama dengan *age_N* maka *flag_same_age* menjadi 5'b10010.

Jika ctr bernilai 0, maka *age_W* akan dibandingkan dengan *age_PE*. Apabila *age_W* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_PE* lebih besar maka nilai *oldest_age_idx* menjadi 0 dan *flag_same_age* menjadi 5'b00000. Apabila *age_W* sama dengan *age_PE* maka *flag_same_age* menjadi 5'b10001.

Setelah komparasi dilakukan maka akan kembali ke proses awal (Gambar 3.1). Jika *oldest_age_idx* tidak bernilai 4, maka proses akan langsung dilanjutkan ke diagram alir B pada Gambar 3.3.



Gambar 3.3 Diagram Alir B

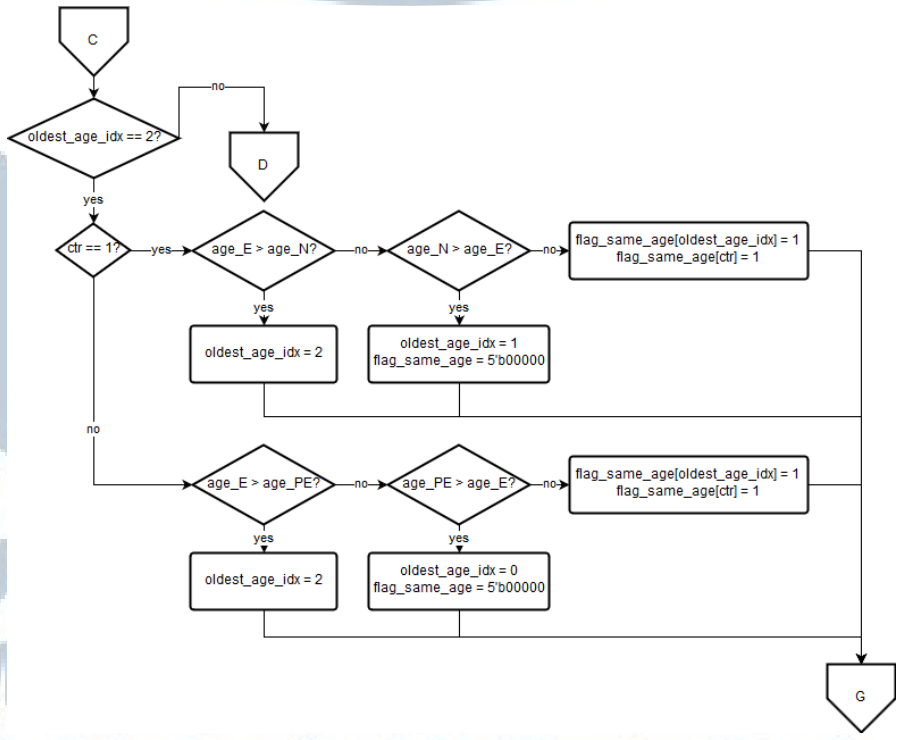
Proses komparasi pada diagram alir B (Gambar 3.3) dimulai jika *oldest_age_idx* bernilai 3, dimana *age* tertinggi saat ini berada di S. Selanjutnya komparasi akan dilakukan dengan mengecek nilai *ctr*.

Jika ctr bernilai 2, maka *age_S* akan dibandingkan dengan *age_E*. Apabila *age_S* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_E* lebih besar maka nilai *oldest_age_idx* menjadi 2 dan *flag_same_age* menjadi 5'b00000. Apabila *age_S* sama dengan *age_E* maka *flag_same_age* menjadi 5'b01100.

Jika ctr bernilai 1, maka *age_S* akan dibandingkan dengan *age_N*. Apabila *age_S* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_N* lebih besar maka nilai *oldest_age_idx* menjadi 1 dan *flag_same_age* menjadi 5'b00000. Apabila *age_S* sama dengan *age_N* maka *flag_same_age* menjadi 5'b01010.

Jika ctr bernilai 0, maka *age_S* akan dibandingkan dengan *age_PE*. Apabila *age_S* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_PE* lebih besar maka nilai *oldest_age_idx* menjadi 0 dan *flag_same_age* menjadi 5'b00000. Apabila *age_S* sama dengan *age_PE* maka *flag_same_age* menjadi 5'b01001.

Setelah komparasi dilakukan maka akan kembali ke proses awal (Gambar 3.1). Jika *oldest_age_idx* tidak bernilai 3, maka akan langsung dilanjutkan ke diagram alir C pada Gambar 3.4.



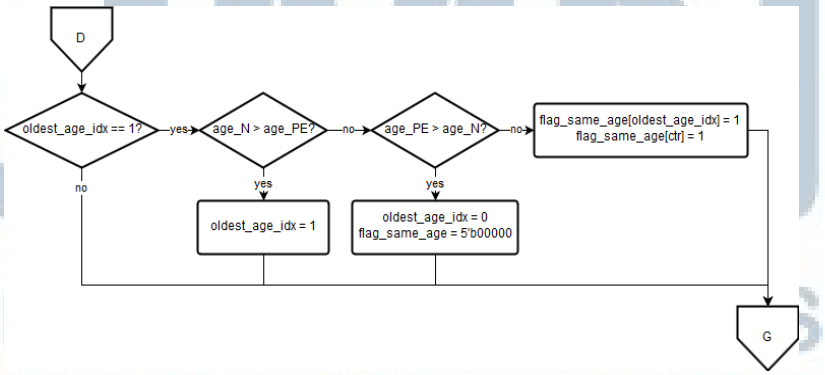
Gambar 3.4 Diagram Alir C

Proses komparasi pada diagram alir C (Gambar 3.4) dimulai jika *oldest_age_idx* bernilai 2, dimana *age* tertinggi saat ini berada di E. Selanjutnya komparasi akan dilakukan dengan mengecek nilai *ctr*.

Jika ctr bernilai 1, maka *age_E* akan dibandingkan dengan *age_N*. Apabila *age_E* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_N* lebih besar maka nilai *oldest_age_idx* menjadi 1 dan *flag_same_age* menjadi 5'b00000. Apabila *age_E* sama dengan *age_N* maka *flag_same_age* menjadi 5'b00110.

Jika ctr bernilai 0, maka *age_E* akan dibandingkan dengan *age_PE*. Apabila *age_E* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_PE* lebih besar maka nilai *oldest_age_idx* menjadi 0 dan *flag_same_age* menjadi 5'b00000. Apabila *age_E* sama dengan *age_PE* maka *flag_same_age* menjadi 5'b00101.

Setelah komparasi dilakukan maka akan kembali ke proses awal (Gambar 3.1). Jika *oldest_age_idx* tidak bernilai 2, maka akan langsung dilanjutkan ke diagram alir D pada Gambar 3.5.



Gambar 3.5 Diagram Alir D

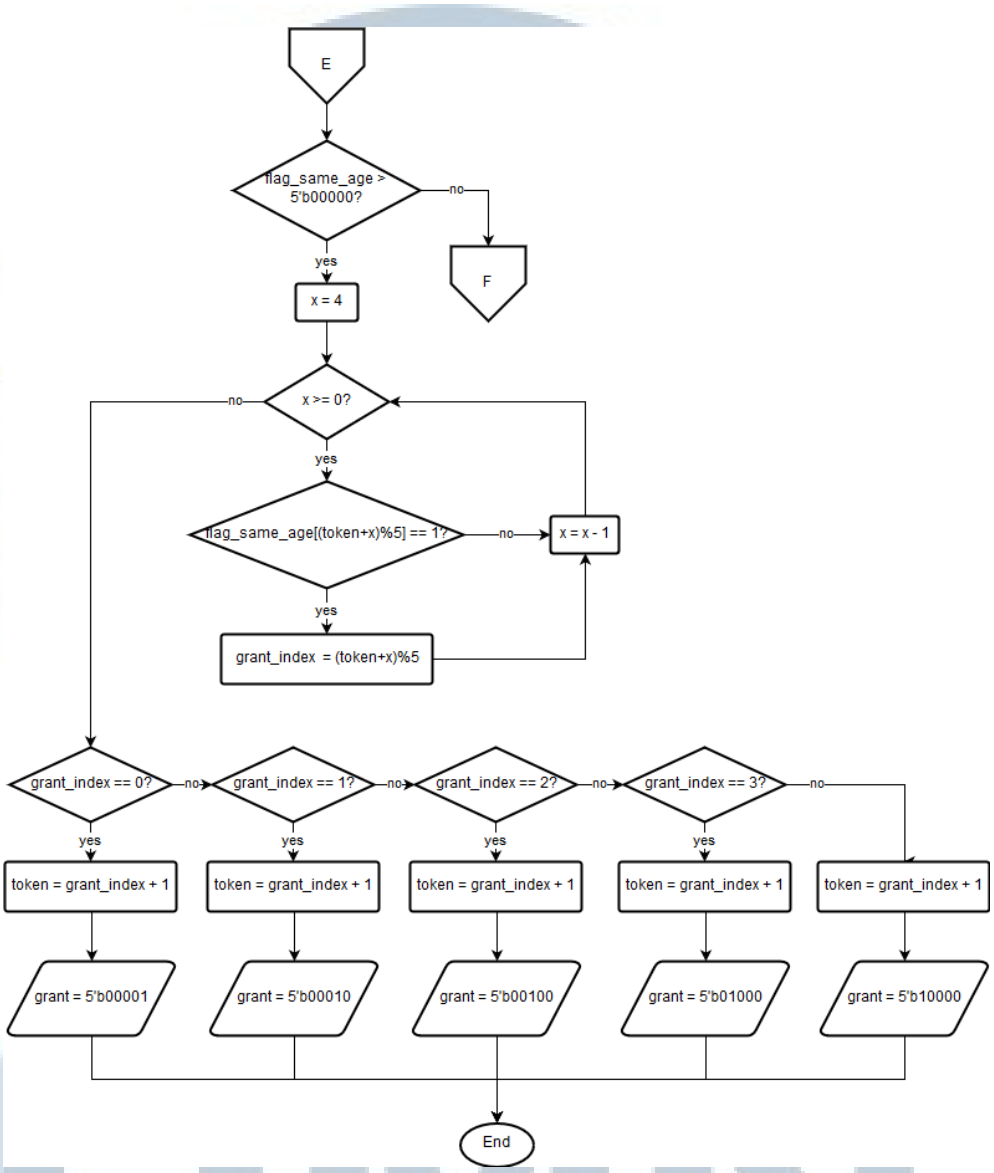
Proses komparasi pada diagram alir D (Gambar 3.5) dimulai jika *oldest_age_idx* bernilai 1, dimana *age* tertinggi saat ini berada di N. Pada bagian ini, *ctr* pasti sudah bernilai 0.

Selanjutnya, *age_N* akan dibandingkan dengan *age_PE*. Apabila *age_N* lebih besar maka nilai *oldest_age_idx* tidak berubah. Apabila *age_PE* lebih besar maka nilai *oldest_age_idx* menjadi 0 dan *flag_same_age* menjadi 5'b00000. Apabila *age_N* sama dengan *age_PE* maka *flag_same_age* menjadi 5'b00011. Setelah komparasi dilakukan maka akan kembali ke proses awal (Gambar 3.1).

Pada proses awal, jika nilai *ctr* lebih kecil dari 0 maka akan dilanjutkan ke proses pengecekan *flag_same_age* yang digambarkan dalam diagram alir E dan F.

Jika nilai *flag_same_age* lebih dari 5'b00000, artinya terdapat dua atau lebih bit *req* dengan nilai *age* tertinggi yang sama. Proses ini kemudian dilanjutkan dengan algoritma round robin, untuk menentukan bit mana yang akan diberikan *grant.* Proses round robin akan dijelaskan pada diagram alir E (Gambar 3.6).

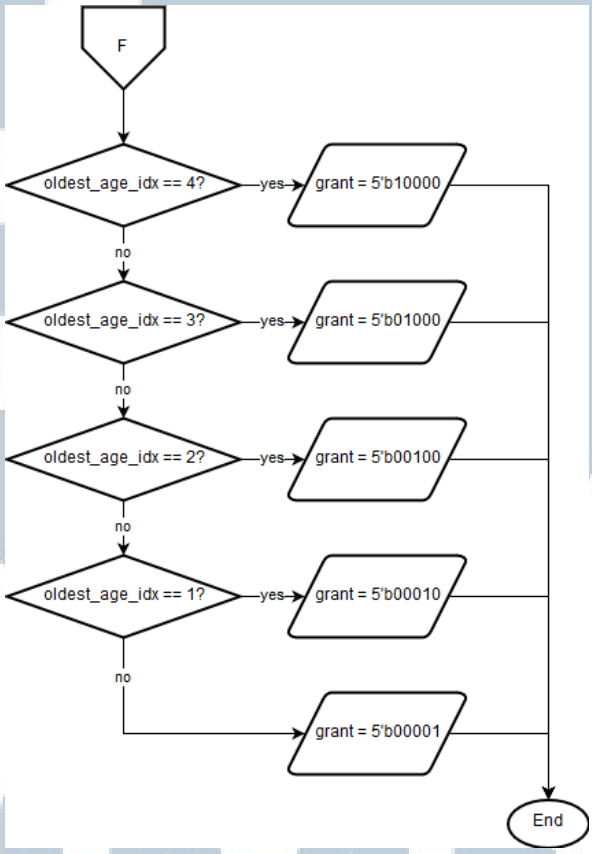
Pada Gambar 3.6, setelah proses komparasi *age*, dilakukan proses pengecekan *flag_same_age*. Jika *flag_same_age* lebih dari 5'b00000, maka akan dijalankan proses round robin yang dimulai dengan mendeklarasikan variabel *x* sebagai *counter*. *Loop* ini berfungsi untuk mencari index *flag_same_age* yang bernilai 1 dan paling dekat dengan *token.* Setelah itu, index tersebut akan disimpan sebagai *grant_index*, yaitu index dari *req* yang diberikan *grant*.



Gambar 3.6 Diagram Alir E

Selanjutnya dilakukan proses pengecekan *grant_index*. Jika *grant_index* bernilai 0, maka *grant* yang diberikan adalah 5'b00001 (PE). Jika *grant_index* bernilai 1, maka *grant* yang diberikan adalah 5'b00010 (N). Jika *grant_index* bernilai 2, maka *grant* yang diberikan adalah 5'b00100 (E). Jika *grant_index* bernilai 3, maka *grant* yang diberikan adalah 5'b01000 (S). Jika *grant_index* adalah 4, maka *grant* yang diberikan adalah 5'b10000 (W). Setelah itu, *token* akan berpindah ke index selanjutnya.

Jika *flag_same_age* bernilai 5’b00000, artinya hanya terdapat 1 bit dari *req* yang memiliki *age* tertinggi, sehingga langsung dapat diberikan *grant* seperti pada diagram alir F (Gambar 3.7).



Gambar 3.7 Diagram Alir F

Pada diagram alir F (Gambar 3.7), dilakukan pengecekan nilai dari *oldest_age_idx* untuk menentukan nilai *grant* seperti pada Tabel 3.1.

Tabel 3.1 Pengecekan Nilai Oldest_age_idx

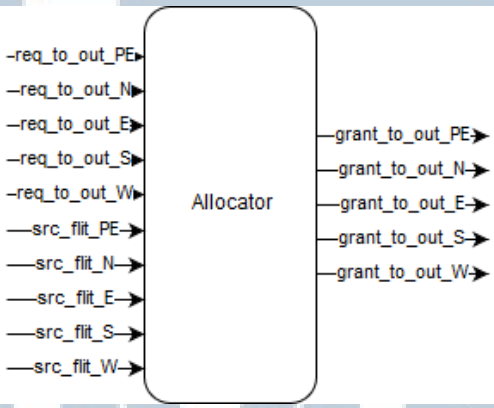
Oldest_age-idx	Grant
4	5’b10000
3	5’b01000
2	5’b00100
1	5’b00010
0	5’b00001

3.3 Perancangan pada Register Transfer Level (RTL)

Pada RTL ditambahkan beberapa input ke allocator dan arbiter. Beberapa RTL yang ditambahkan parameternya yaitu *xbar_allocator* dan *arbiter_xbar*.

Pada *xbar_allocator* (Gambar 3.8), ditambahkan beberapa input, yaitu:

1. *src_flit_PE*, source address untuk flit dari yang berasal dari PE
2. *src_flit_N*, source address untuk flit dari yang berasal dari N
3. *src_flit_E*, source address untuk flit yang berasal dari E
4. *src_flit_S*, source address untuk flit yang berasal dari S
5. *src_flit_W*, source address untuk flit yang berasal dari W



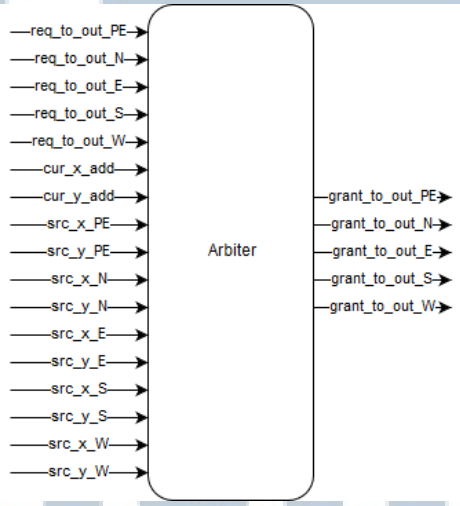
Gambar 3.8 Modified Allocator

Input ini akan dibaca alamat X dan Y-nya lalu kemudian diteruskan ke arbiter. Selain itu, juga ditambahkan variabel untuk menyimpan alamat X dan Y dari *current router* untuk dikirim juga ke arbiter.

Pada *arbiter_xbar* (Gambar 3.9), ditambahkan input-input dari *xbar_allocator*, yaitu:

1. *current_x_address*, alamat X dari router sekarang
2. *current_y_address*, alamat Y dari router sekarang

3. *source_x_address_from_PE*, alamat X untuk flit yang berasal dari PE
4. *source_y_address_from_PE*, alamat Y untuk flit yang berasal dari PE
5. *source_x_address_from_N*, alamat X untuk flit yang berasal dari N
6. *source_y_address_from_N*, alamat Y untuk flit yang berasal dari N
7. *source_x_address_from_E*, alamat X untuk flit yang berasal dari E
8. *source_y_address_from_E*, alamat Y untuk flit yang berasal dari E
9. *source_x_address_from_S*, alamat X untuk flit yang berasal dari S
10. *source_y_address_from_S*, alamat Y untuk flit yang berasal dari S
11. *source_x_address_from_W*, alamat X untuk flit yang berasal dari W
12. *source_y_address_from_W*, alamat Y untuk flit yang berasal dari W



Gambar 3.9 Modified Arbiter

Semua input ini digunakan dalam perhitungan *age* dari tiap flit. A*ge* didapatkan dengan menghitung selisih alamat X dari router sekarang dan router source, lalu selisih alamat Y dari router sekarang dan router source, kemudian kedua hasil dijumlahkan.