TUGAS AKHIR - TE 141599

ANALISIS UNJUK KERJA *FLOW CONTROL* PADA *NETWORK ON CHIP* DALAM BEBERAPA KONDISI JARINGAN

M Hizrian Hizburrahman
NRP 2211 100 128

Dosen Pembimbing
Dr. Istas Pratomo, ST., MT.
Ir. Djoko Suprajitno Rahardjo, MT.

JURUSAN TEKNIK ELEKTRO
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya 2015

FINAL PROJECT - TE 141599

PERFORMANCE ANALYSIS OF FLOW CONTROL ON NETWORK ON CHIP IN SEVERAL NETWORK CONDITION

M Hizrian Hizburrahman
NRP 2211 100 128

Supervisors
Dr. Istas Pratomo, ST., MT.
Ir. Djoko Suprajitno Rahardjo, MT.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Industrial Technology
Sepuluh Nopember Institute of Technology
Surabaya 2015

ANALISIS UNJUK KERJA *FLOW CONTROL* PADA *NETWORK ON CHIP* DALAM BEBERAPA KONDISI JARINGAN

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Telekomunikasi Multimedia
Jurusan Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui :

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Istas Pratomo, ST., MT.	**Ir. Djoko Suprajitno R, MT.**
NIP. 197903252003121001	NIP. 195506221987011001

SURABAYA
JUNI, 2015

ANALISIS UNJUK KERJA *FLOW CONTROL* PADA *NETWORK ON CHIP* DALAM BEBERAPA KONDISI JARINGAN

M Hizrian Hizburrahman
2211 100 128

Dosen Pembimbing I : Dr. Istas Pratomo, ST., MT.
Dosen Pembimbing II : Ir. Djoko Suprajitno Rahardjo, MT.

ABSTRAK

Network on Chip (NoC) merupakan solusi yang digunakan pada *System on Chip* (SoC) sebagai pengganti teknik *shared bus* dan *direct point – to – point* dimana meningkatkan efisiensi dan mengatasi *delay* interkoneksi. Performansi jaringan dipengaruhi oleh parameter desain sehingga menentukan parameter desain yang tidak tepat dan dapat menimbulkan permasalahan seperti *congestion* dan saturasi jaringan yang menyebabkan paket hilang. *Congestion* dapat diatasi dengan menggunakan *flow control* yang tepat.

Pada tugas akhir ini dilakukan analisis *flow control* pada NoC yaitu Stall / Go, Ack / Nack, dan *Dynamic Multi Level* terhadap *flow control default* Noxim. Keempat metode tersebut diterapkan pada jaringan yang tersaturasi dengan kombinasi ukuran *buffer* & PIR, PIR & ukuran paket untuk dianalisis pengaruhnya terhadap titik saturasi dan *throughput* maksimum. *Flow control* tersebut juga diterapkan pada jaringan dengan parameter desain yang berubah untuk mendapatkan *flow control* terbaik dalam meningkatkan performansi jaringan.

Dari data yang diperoleh, pada jaringan yang menerapkan *flow control* Stall / Go tidak mengalami saturasi *throughput*. Untuk perubahan ukuran jaringan, PIR, ukuran paket dan ukuran buffer, *Flow control* Stall / Go memberikan peningkatan *throughput* sebesar 21.08 %, 65.33%, 151%, dan 13.37%, penurunan *delay* sebesar 407.85, 606.03, 1631.95, 322.59 cycles, penurunan penggunaan daya sebesar 68.67%, 61.33%, 49.93%, 68.22%

Kata Kunci : *System on Chip, Network on Chip, Flow Control, Congestion*

PERFORMANCE ANALYSIS OF *FLOW CONTROL* ON *NETWORK ON CHIP* IN SEVERAL NETWORK CONDITION

M Hizrian Hizburrahman
2211 100 128

Supervisor I : Dr. Istas Pratomo, ST., MT.
Supervisor II : Ir. Djoko Suprajitno R, MT.

ABSTRACT

Network on Chip (NoC) is a solution for System on Chip which replaces shared bus and direct point – to – point. NoC increases efficiency and decreases interconnection *delay* in SoC. Network performancy can be affected by design's parameter so choosing wrong design's parameter will lead to *congestion* and network's saturation that results to packetloss. *Congestion* can be reduce by using *flow control*.

In this final work, Stall / Go, Ack / Nack, Dynamic Multi Level *flow control* are compare with Default *flow control* and will be analyzed in network with combination of design parameter. First combination is *Buffer* Size and *Packet injection rate*, second combination is *Packet injection rate* and Packet Size. In this condition, impact from *flow control* on saturation point and maximum *throughput* will be analyzed. In this final work, the best *flow control* that increase network's performance also will be choosen.

From the data results, it is obtained that network which using Stall/Go *flow control* is not saturate. In a condition where network's size, PIR, packet's size, and buffer's size are changing Stall / Go *flow control* increase *throughput* 21.08 %, 65.33%, 151%, and 13.37%, decrease *delay* 407.85, 606.03, 1631.95, 322.59 cycles, decrease power consumption 68.67%, 61.33%, 49.93%, and 68.22%.

Keyword : *System on Chip, Network on Chip, Flow Control, Congestion*

KATA PENGANTAR

Alhamdulillah, puji dan syukur penulis panjatkan kehadirat Allah SWT serta tak lupa shalawat dan salam dihaturkan kepada Nabi Muhammad SAW, keluarga, sahabat dan pengikut beliau hingga akhir zaman. Karena atas rahmat dan karunia – Nya penulis dapat menyelesaikan penulisan buku Tugas Akhir dengan judul :

"ANALISIS UNJUK KERJA METODE *FLOW CONTROL* PADA *NETWORK ON CHIP* DALAM BERBAGAI KONDISI JARINGAN "

Tugas akhir ini disusun untuk memenuhi sebagian persyaratan dalam menyelesaikan pendidikan Strata – 1 pada Bidang Studi Telekomunikasi Multimedia, Jurusan Teknik Elektro, Fakultas Teknologi Industri, Institut Teknologi Sepuluh nopember Surabaya.

Penulis menyadari bahwa dalam penulisan Tugas Akhir ini banyak mengalami kendala, namun berkat bantuan, bimbingan dan kerjasama dari berbagai pihak segala kendala dapat teratasi. Untuk itu penulis menyampaikan banyak terima kasih kepada :

1. Kedua orang tua, Bapak Agus Cahyono dan Ibu Catur Sri Rejeki Handayani yang selalu memberikan dukungan, semangat, dan doa kepada penulis.
2. Bapak Istas Pratomo dan Bapak Djoko Suprajitno selaku Dosen Pembimbing atas segala bantuan, perhatian, dan arahan selama pengerjaan Tugas Akhir ini.
3. Bapak Endroyono selaku Koordinator Bidang Studi Telekomunikasi Multimedia Jurusan Teknik Elektro ITS.
4. Bapak Tri Arief Sardjono selaku Ketua Jurusan Teknik Elektro ITS.
5. Bapak dan Ibu dosen bidang studi Telekomunikasi Multimedia, Teknik Elektro ITS
6. Semua rekan – rekan yang turut membantu dan memberikan semangat serta kecerian dalam pengerjaan tugas akhir ini

Penulis menyadari bahwa pada penyusunan laporan tugas akhir ini masih terdapat kekurangan-kekurangan karena keterbatasan kemampuan yang penulis miliki, walaupun demikian penulis berharap tugas akhir ini dapat bermanfaat bagi yang membutuhkannya.

Surabaya, Mei 2015

Penulis

DAFTAR ISI

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Semakin dengan berkembangnya teknologi, ukuran dari perangkat elektronik semakin kecil. Hal ini dikarenakan perkembangan dari teknologi *Multiprocesso* dan *System on Chip* (SoC). Dimana perangkat elektronik saat ini menggunakan *chip* yang lebih dari satu. *Chip – chip* yang ada tersebut dapat saling berkomunikasi satu sama lain dengan menggunakan teknik komunikasi seperti *shared bus, direct point – to – point* dan juga *Network on Chip* (NoC).

Dimana *Direct point – to – point* menggunakan kabel yang terdedikasi untuk PE tertentu tanpa adanya arbitrasi terpusat. Penggunaan kedua teknologi tersebut memiliki kelebihan yaitu sederhana dan mudah dalam pengaturannya. Tetapi seiring berkembanganya teknologi silikon yang membuat ukuran dari *node* yang ada semakin besar, menimbulkan permasalahan seperti terjadinya *bottleneck* pada *shared bus* dan meningkatnya kompleksitas dan *Delay* pada *Direct point – to – point.* Sehingga untuk dapat mengatasi permasalahan tersebut dan untuk meningkatkan efisiensi digunakan *Network on Chip (NoC)* sebagai solusi untuk komunikasi pada infrastruktur pada SoC yang dapat memberikan reliabilitas tinggi dan *delay* interkoneksi yang rendah [1].

Dalam mendesain NoC harus dapat mencapai performansi yang sesuai QoS yaitu seperti *Throughput* tinggi, *Delay* rendah, penggunaan sumber daya yang rendah dan area yang kecil [2]. Sehingga diperlukan adanya *trade – off* antara parameter performa jaringan dan parameter desain untuk dapat mencapai performa yang diharapkan. Parameter desain pada NoC berupa ukuran jaringan, *flow control*, ukuran paket, ukuran *buffer*, *packet injection rate* dan Elemen desain seperti Topologi, algoritma *routing*, model trafik, teknik *switching*.

Sebelum dilakukan proses fabrikasi dilakukan proses pendesainan terlebih dahulu untuk dapat memperoleh jaringan dengan performa yang diharapkan. Akan tetapi apabila penentuan parameter desain tidak sesuai dapat menimbulkan permasalahan seperti *congestion* dan saturasi.

Congestion terjadi karena adanya *overload* pada *buffer* penerima yang dikarenakan adanya ketidak sesuaian antara ukuran

buffer dan ukuran paket yang dikirimkan serta informasi mengenai *buffer* penerima yang tidak diketahui oleh node pengirim. *Congestion* dapat menyebabkan paket yang diterima akan di drop yang menyebabkan *error* dan menurunnya performa jaringan. Selain itu dapat pula terjadinya kondisi saturasi dimana *throughput* pada jairngan mencapai nilai maksimum meskipun parameter desain lainnya berubah nilanya. Memiliki titik saturasi yang rendah dan *throughput* maksimal yang rendah dapat menjadi permasalahan lainnya.

Dengan Menggunakan *flow control* dimana merupakan salah satu teknik yang dapat digunakan untuk mengatur transmisi data untuk mengatasi *congestion*[4]. *Flow Control* dapat didesain untuk mencegah terjadinya penggunaan antian yang berlebihan, mengatur pengalokasian *buffer* dan *link* pada setiap *router*, menentukan kapan *buffer* dan *link* akan digunakan, serta bagaimana pembagian sumber daya yang digunakan secara bersama.

Oleh karena itu pemilihan *flow control* yang tepat untuk kondisi jaringan tertentu menjadi sangat penting untuk dapat mengatasi permasalah tersebut. Sehingga dapat meningkatkan performa jaringan untuk mencapai nilai yang diinginkan. Pemilihan *flow control* juga harus memperhatikan pengaruhnya terhadap perubahan dari elemen desain seperti penambahan overhead, area, wiring yang dapat mempengaruhi performa jaringan.

Pada tugas akhir ini akan dilakukan analisis terhadap tiga buah metode *flow control* yang ada pada NoC yaitu STALL / GO, ACK / NACK, serta Dynamic Multi Level *flow control* yang menggunakan konsep *buffer fluidity control*. Dari ketiga metode *flow control* tersebut kemudian akan dianalisis pengaruhnya terhadap kondisi saturasi jaringan dan peningkatan performa jaringan. Sehingga dapat ditentukan metode *flow control* yang memberikan hasil paling optimal.

1.2 Perumusan Masalah

Beberapa permasalahan yang mendasari Tugas Akhir ini adalah

1. Bagaimana pengaruh dari implementasi *flow control* pada NoC terhadap titik saturasi pada jaringan ?
2. *Flow control* manakah yang memberikan pengaruh paling besar dalam kondisi saturasi
3. Bagaimana pengaruh dari implementasi *flow control* pada NoC terhadap performa jaringan dengan penggunaan parameter desain yang berbeda – beda ?

4. *Flow control* manakah yang paling optimal dalam memberikan peningkatan performa jaringan terbaik pada parameter performansi *throughput*, *delay*, dan daya yang digunakan ?

1.3 Batasan Masalah

Agar penelitian tidak menyimpang dari permasalahan maka penulis membatasi masalah sebagai berikut :

1. Desain titik saturasi dan *throughput* maksimum saat kondisi saturasi pada jaringan.
2. Implementasi teknik STALL / GO, ACK/NACK, DyML *Flow Control* pada kondisi saturasi jaringan.
3. Pengukuran dan perbandingan parameter performansi jaringan *throughput* dan titik saturasi yang telah diimplementasikan berbagai teknik *flow control* pada kondisi saturasi jaringan.
4. Desain kondisi jaringan dengan menggunakan XY routing algorithm dengan variabel parameter desain berupa Ukuran Jaringan, *Packet injection rate*, Ukuran Paket, Ukuran *Buffer* dengan model trafik transpose1.
5. Implementasi teknik STALL / GO, ACK/NACK, DyML *Flow Control* pada kondisi jaringan hasil desain.
6. Pengukuran dan perbandingan parameter performansi jaringan *throughput*, *delay*, dan daya, yang telah diimplementasikan berbagai teknik *flow control* pada kondisi jaringan hasil desain.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah untuk :

1. Menganalisis pengaruh *flow control* terhadap kondisi saturasi pada jaringan
2. Menganalisis pengaruh *flow control* terhadap performa jaringan dengan parameter desain tertentu
3. Menentukan *flow control* terbaik dalam kondisi saturasi pada jaringan
4. Menentukan *flow control* terbaik dalam meningkatkan performa jaringan

1.5 Metodologi

Metode Penelitian yang digunakan pada tugas akhir ini terbagi menjadi tujuh tahap sebagai berikut:

1.5.1 Studi Literatur

Pada tahapan studi literatur dilakukan pencarian amteri yang berkaitan dengan pokok bahasan pada Tugas Akhir ini. Referensi dapat bersumber dari jurnal maupun buku – buku yang ada. Dimana hal – hal yang dipelajari lebih dalam yaitu mengenai :

- Konsep dari Arsitektur *Network on Chip*
- Konsep dari Routing Algorithm pada *Network on Chip*
- Konsep Parameter Desain dan Parameter Performa Jaringan
- Konsep dari STALL / GO *Flow Control*
- Konsep dari ACK / NACK *Flow Control*
- Konsep dari Dynamic Multi Level *Flow Control*
- Bahasa Pemrograman C++
- Penggunaan Software Simulasi Noxim

1.5.2 Desain Jaringan

Pada tahapan desain jaringan terbagi menjadi dua tahapan, yaitu desain jaringan untuk kondisi jaringan yang mengalami saturasi dan desain jaringan dengan variabel parameter desain yang memiliki nilai tertentu.

Pada desain jaringan kondisi saturasi, nilai dari *throughput* jaringan akan selalu bernilai maksimum pada nilai tertentu. Pada tahapan ini digunakan dua buah kombinasi parameter desain yaitu :

- Kombinasi *Buffer* Size dan *Packet injection rate*
- Kombinasi *Packet injection rate* dan Ukuran Paket

Dimana Routing Algorithm yang digunakan adalah XY Routing Algorithm dengan model trafik transpose dengan ukuran jaringan adalah 5 x 5 mesh.

Pada desain jaringan dengan variabel parameter yang berubah – ubah, parameter performa jaringan yang digunakan yaitu *Throughput*, *Delay*, Power.

Desain yang akan berubah nilainya untuk mengetahui pengaruhnya terhadap performa jaringan adalah Ukuran Jaringan, *Packet injection rate*, Ukuran Paket, Ukuran *Buffer*

Dimana saat Ukuran Jaringan merupakan parameter desain yang berubah, maka parameter desain lainnya akan bernilai tetap. Adapun Routing Algorithm yang digunakan adalah XY routing

algorithm. Dimana model trafik transpose dan switching algorithm yang digunakan berupa random.

1.5.3 Penulisan Algoritma *Flow Control*

Penulisan algoritma untuk masing – masing metode *flow control* yaitu STALL / GO *flow control*, ACK / NACK *flow control* dan DyML *Flow Control* akan dilakukan pada tahapan ini.

1.5.4 Implementasi *Flow Control*

Pada tahapan ini, proses pertama adalah dengan melakukan pengkodean algoritma untuk masing – masing *flow control* ke dalam bahasa C dan C++ yang akan digunakan oleh Noxim Simulator.

Kemudian, masing – masing *flow control* akan diimplementasikan untuk kedua desain jaringan yang telah dibuat pada tahapan sebelumnya sehingga dapat dilakukan pengambilan data pada tahapan berikutnya.

1.5.5 Pengambilan Data

Pada tahapan ini dilakukan pengambilan data parameter performansi jaringan pada kedua desain jaringan yang telah dibuat. Dimana parameter performansi jaringan yang akan diambil datanya adalah *Throughput*, *Delay* dan Daya.

Untuk kondisi jaringan saturasi, akan dilakukan pengambilan data berupa titik saat terjadi saturasi dan nilai *throughput* maksimum untuk kedua skenario kombinasi yang telah ditentukan dimana dengan Default *Flow Control*, menggunakan STALL / GO, ACK/NACK, Dynamic Multi Level

Untuk kondisi kedua akan dilakukan pengambilan data parameter performansi berupa *Throughput*, *Delay* dan Daya.untuk masing – masing variabel parameter desain yang telah ditentukan sebelumnya yaitu Ukuran Jaringan, *Packet injection rate*, Ukuran Paket, Ukuran *Buffer*. Dimana akan dilakukan pengambilan data untuk mengetahui pengaruh *flow control* terhadap performa jaringan dengan Default *Flow Control*, menggunakan STALL / GO, ACK/NACK, Dynamic Multi Level.

1.5.6 Analisis Data

Pada tahapan ini akan dilakukan analisis dari hasil data yang telah diperoleh dimana untuk kondisi saturasi akan dilakukan analisis

mengenai pengaruh yang diberikan oleh masing – masing *flow control* terhadap titik saturasi dan *throughput* maksimum yang terjadi pada jaringan

Selain itu, akan dilakukan pula analisis pengaruh dari *flow control* terhadap peningkatan parameter performansi jaringan *throughput*, *delay* dan daya yang digunakan pada jaringan.

1.5.7 Penarikan Kesimpulan

Kemudian dapat dilakukan penarikan kesimpulan mengenai *flow control* yang memberikan hasil optimal pada kondisi saturasi dan *flow control* yang memberikan perubahan parameter performansi jaringan yang optimal

1.6 Sistematika Pembahasan

Pembahasan dalam tugas akhir ini akan dibagi dalam lima bab dengan sistematika sebagai berikut:

BAB 1 PENDAHULUAN

Pada bab ini akan diuraikan mengenai latar belakang, permasalahan, tujuan penelitian, metodologi penelitian, sistematika laporan, dan relevansi.

BAB II TINJAUAN PUSTAKA

Pada bab ini akan dijelaskan tentang tinjauan pustaka yang akan membahas tentang Arsitektur, komponen, parameter – parameter yang ada pada *Network on Chip*, *flow control*, baik itu *STALL / GO, ACK / NACK, DyML Flow control*.

BAB III PERANCANGAN DAN IMPLEMENTASI *FLOW CONTROL*

Pada bab ini akan dijelaskan tentang parameter – parameter yang digunakan dalam mendesain jaringan, dan juga pengimplementasian *flow control* pada jaringan berdasarkan teori pada Bab II serta melakukan pengujian unjuk kerja sistem.

BAB IV HASIL DAN ANALISIS DATA

Pada bab ini akan ditampilkan hasil pengujian, kemudian dilakukan analisis dari data yang telah diperoleh berdasarkan rumusan masalah.

BAB V KESIMPULAN DAN SARAN

Pada bab ini berisi tentang kesimpulan, dan saran berdasarkan yang telah dilakukan dalam pengerjaan tugas akhir ini.

1.7 Relevansi

Hasil yang didapat dari tugas akhir ini diharapkan dapat memberi manfaat sebagai berikut :

1. Memberikan analisis pengaruh *flow control* untuk kondisi saturasi pada jaringan dan pengaruh *flow control* terhadap performa jaringan dengan parameter desain tertentu.
2. Memberikan metode *flow control* terbaik untuk kondisi saturasi pada jaringan
3. Memberikan metode *flow control* yang memberikan pengaruh terbaik pada jaringan dengan parameter desain tertentu.

BAB 2
NETWORK ON CHIP DAN *FLOW CONTROL*

2.1 Sistem pada Chip (*System on Chip*)

Semakin berkembangnya teknologi, ukuran dari perangkat elektronik semakin kecil dengan fitur atau kemampuan yang canggih. Hal ini dikarenakan adanya *System on Chip* (SoC). Pada SoC *chip* yang ada disebut sebagai *Intellectual Property* (IP) atau *core*. Setiap *core* tersebut dapat berkomunikasi satu sama lain dengan menggunakan teknik seperti *direct point – to – point, shared bus*, hingga *network on chip (NoC).*

2.1.1 Direct Point – to - Point

Teknologi *Direct point – to – point* merupakan teknik komunikasi yang digunakan pertama kali pada *system on chip*.Setiap sumberdaya atau *core* yang ada dapat saling berkomunikasi satu sama lain secara langsung melalui kabel yang menghubungkan *core – core* tersebut. Pada teknologi ini tidak membutuhkan prioritas atau *arbitration unit*. *System on chip* yang memiliki banyak *core* akan menimbulkan *delay* dan kebutuhan akan jumlah pin untuk setiap *core* meningkat yang membuat pengkabelan menjadi rumit. *Direct point – to point* memiliki skalabilitas yang rendah dan kompleksitas yang tinggi[5].

2.1.2 Shared Bus

Teknologi shared bus digunakan pada SoC sebagai komunikasi inter-*core* yang menggantikan *direct point – to – point*. Pada *shared bus* digunakan *interface* untuk menghubungkan bus dengan *core*. Pada teknologi ini, komunikasi dan *Congestion* diatur oleh *bus arbiter*. *Shared bus* membutuhkan *input* dan *output* pin yang lebih sedikit jika dibandingkan dengan *direct point – to – point* sehingga area untuk pengkabelan dan biaya yang diperlukan lebih murah. Terdapat beberapa jenis dari bus yang ada yaitu seperti hierarki, segmentasi, dan *pipelined* buses. Kekurangan yang dimiliki oleh teknologi *shared bus* adalah proses transmisi data menjadi lambat dikarenakan adanya *Congestion* dan arbitration. Selain itu dari sisi skalabilitas, *shared bus* tidak efisiensi[5].

.

2.1.3 Jaringan pada Chip (*Network on Chip*)

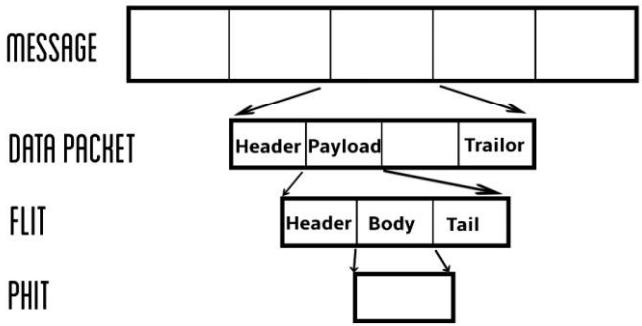
Kedua teknologi yang digunakan sebelumnya yaitu *direct point – to – point dan shared bus* merupakan teknologi yang sederhana dan mudah untuk diatur. Tetapi seiring dengan berkembangnya teknologi *silicon* yang membuat jumlah *core* semakin meningkat, beberapa permasalahan muncul pada teknologi *shared bus* dan *direct point – to – point*. Seperti terjadinya *bottleneck* pada *shared bus* yang meningkatkan *delay* dan jumlah pin yang semakin banyak yang membuat pengkabelan semakin kompleks pada *direct point – to – point*. Sehingga teknologi *shared bus dan direct point – to – point* tidak memungkinkan untuk mendukung kebutuhan komunikasi antar *core* yang ada pada SoC. Sehingga untuk dapat mendukung kebutuhan tersebut digunakan teknologi *Network on Chip*[5].

Network on Chip memberikan infrastruktur yang fleksibel untuk mengatasi penurunan performansi yang disebabkan karena perubahan pada lingkungan atau kebutuhan akan aplikasi tertentu. *Network on Chip* mengadopsi arsitektur dari jaringan komputer untuk mengirimkan dan menerima paket antar *core* atau *processing element* (PE). *Network on Chip* menggunakan *switch* atau *router* untuk meneruskan paket dari pengirim ke penerima dengan menggunakan algoritma routing. Pada *Network on Chip*, semua pengkabelan dapat berupa pipelined sehingga performansi *local* tidak akan menurun saat ukuran jaringan meningkat.

2.2 Struktur Pesan pada *Network on Chip*

Proses komunikasi pada *Network on Chip* melibatkan beberapa bentuk dari pesan. Dimana pesan merupakan sebuah informasi atau data yang ditransmisikan dari PE sumber ke PE tujuan. Pesan didefinisikan pada layer aplikasi. Sebuah pesan dapat memiliki panjang yang tetap atau tidak tetap yang bergantung pada kebutuhan. Pesan tersebut akan bergerak pada jaringan dalam berbagai bentuk seperti paket, flit dan phit[5].

Proses packetization membuat pesan dibagi menjadi beberapa paket dimana paket dari pesan yang sama independen dengan paket lainnya. Setiap paket memilki informasi yang cukup untuk bergerak pada jaringan. Pada umumnya paket memiliki tiga bagian yaitu *header*, *payload* atau *body* dan *trailer*. *Header* berisi informasi routing dan informasi control seperti alamat sumber dan alamat tujuan dan atau bahkan berisi semua rute untuk transmisi *data*. *Payload* berisi data yang



Gambar 2.1 Pemecahan Pesan pada *Network on Chip*

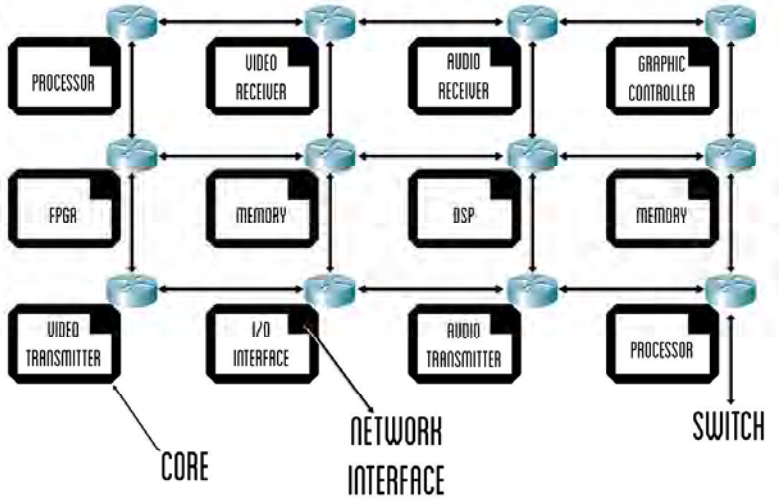
sebenarnya atau informasi yang akan dikirimkan dan trailer mengindikasikan akhir dari paket.

Gambar 2.1 menampilkan konsep pemecahan pesan pada *Network on Chip*[5]. Sebuah paket dapat dibagi menjadi komponen yang lebih kecil lagi yang disebut sebagai flit atau *flow control* digits melalui *flitization*. Flit terdiri dari tiga bagian yaitu *header* flit, *body* flit dan *tail* flit. Ukuran dari setiap flit selalu tetap. Flit biasa digunakan pada jaringan yang menggunakan *buffer* atau router yang berukuran kecil[5].

Flit dapat dibagi menjadi unit yang lebih kecil yang disebut sebagai phit atau *physical transfer digits*. Phit bergerak melewati kanal yang ada diantara *switch* pada jaringan sebagai satu unit. Phit juga dapat dikatakan sebagai lebar dari kanal atau dapat pula diindikasikan sebagai jumlah kabel yang diperlukan untuk mentransmisikan data antar router pada jaringan. Ukuran dari phit dapat sama dengan ukuran flit dapat pula berbeda.

2.3 Arsitektur *Network on Chip*

Arsitektur dari NoC pada umumnya direpresentasikan sebagai kombinasi dari beberapa komponen hardware dan protocol komunikasi atau komponen jaringan yang menentukan arsitektur komunikasi. Dimana sumberdaya yang terintegrasi pada jaringan yang berupa komponen hardware dapat berupa processor, DSP, *Field Programmable Gate Array* (FPGA), memory, atau I/O. Sedangkan komponen jaringan berupa switch/router, kanal, dan *Resource Network Interface* (RNI). Komponen jaringan memberikan layanan komunikasi untuk semua



Gambar 2.2 Arsitektur *Network on Chip* secara Umum

system yang ada. Sumberdaya atau *Processing Elements* (PE) terhubung dengan *switch* melalui RNI sedangkan switch dapat terhubung dengan switch lainnya dengan menggunakan sebuah kanal yang merupakan dua arah sehingga dapat saling berkomunikasi satu sama lain dengan mengirimkan sebuah pesan. Gambar 2.2 menunjukkan arsitektur dari *Network on Chip* secara umum[3]

2.3.1 Resources

Sumber daya atau *Procesing Element* (PE) pada *network on Chip* dapat berupa processor, Field Programmable Gate Array (FPGA), Amplifier, ADC, DSP, memory, graphic controller, RF unit, dan I/O controller.

2.3.2 RNI

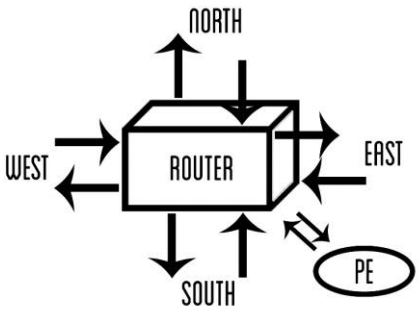
RNI atau *Resource Network Interface* digunakan untuk menghubungkan sebuah PE dengan sebuah switch/router pada *Network on Chip*. Sehingga setiap PE dapat mentransmisikan pesan ke router yang ada. RNI memiliki dua bagian yaitu *Resource Dependent Part* dan *Resource Independent Part*[5].

Resource Independent Part didesain sedemikian rupa sehingga dapat beroperasi layaknya sebuah switch lain yang terhubung dengan switch milik PE tersebut. *Resource Dependent Part* dari RNI memiliki beberapa fungsi yaitu *flitization* atau perubahan paket menjadi flit, *deflitization* atau perubahan dari flit menjadi paket kembali dan menerapkan proses encoding. RNI memiliki dua layer OSI yang saling independent yaitu layer sesi dan layer transport dimana tugas dan fungsi dari kedua layer tersebut berbeda dengan tugasnya pada OSI. Dimana layer sesi berlaku sebagai medium yang efektif untuk menghubungkan PE dan layer transport menjamin proses transmisi data berlangsung dengan baik dan mengoperasikan network interface pada arsitektur komunikasi. Layer transport juga menawarkan layanan komunikasi untuk layer sesi[5].

2.3.3 Switch / Router

Router yang digunakan pada *Network on Chip* merupakan sebuah *switch* yang memiliki peranan penting seperti infrastruktur jaringan lainnya dan *backbone* dari *Network on Chip*[2]. Tugas utama dari *router* adalah untuk mentransmisikan pesan yang diterima ke PE tujuan apabila router terhubung secara langsung dengan PE tujuan. Jika tidak maka *router* akan meneruskan pesan tersebut ke *router* lainnya.

Pada umumnya router yang digunakan pada *Network on Chip* terdiri dari lima buah *port* yaitu *North East, South, West* dan *local port* seperti ditunjukkan pada Gambar 2.3. Empat buah port digunakan untuk menghubungkan router yang satu dengan router lainnya. Sedangkan *local port* digunakan untuk menghubungkan router dengan PE[2].



Gambar 2.3 Model Router yang digunakan Pada *Network on Chip*.

Router memiliki *buffer* dimana *buffer* berfungsi sebagai penyimanan data yang sementara. Terdapat pula *control logic* untuk dapat melakukan fungsi arbitrasi. Koneksi dan atau konfigurasi antar port – port yang ada pada router dilakukan oleh *central crosspoint matrix*.

2.4 Elemen Desain *Network on Chip*

Dalam mendesain sebuah jaringan pada NoC, elemen desain yang penting yaitu Topologi, Teknik Switching dan Algoritma *Routing*.Parameter desain yang penting yaitu ukuran jaringan, ukuran paket, *packet injection rate*, ukuran paket, dan ukuran *buffer*. Performansi dari NoC bergantung pada elemen dan parameter desain tersebut.

Permasalahan utama dalam mendesain NoC adalah bagaimana menghubungkan setiap PE sehingga setiap PE dapat saling berkomunikasi satu sama lain dengan performansi yang diharapkan. Layout fisik dan koneksi antar *node* dan kanal pada jaringan merupakan topologi. Sedangkan arsitektur komunikasi menentukan bagaimana setiap komponen dapat saling berkomunikasi. Algoritma routing dan teknik switching digunakan untuk menentukan rute mana yang akan dilalui oleh pesan untuk mencapai node tujuan[2]. Tujuan dari algoritma routing adalah untuk mendistribusikan *traffic* secara merata pada rute yang disediakan oleh topologi jaringan sehingga dapat menghindari hotspot dan meminimalisir terjadinya *Congestion* sehingga dapat meningkatkan performa jaringan.

Dalam mendesain NoC harus diperhatikan pula QoS yang diharapkan. Batas minimal dari *throughput*, *Delay*, dan jitter harus tercapai untuk dapat memenuhi kebutuhan dari suatu aplikasi. Hampir semua elemen dan parameter desain mempengaruhi performa dari jaringan[2].

2.4.1 Topologi

Salah satu elemen desain yang penting adalah topologi jaringan dimana memberikan pengaruh signifikan terhadap performa dari NoC dikarenakan topologi menentukan jarak antar node yang saling terhubung[3]. Topologi dapat didefinisikan sebagai layout fisik yang mendeskripsikan bagaimana router pada NoC dapat terhubung satu sama lain. Persyaratan utama dari sebuah topologi adalah skalabilitas tinggi

dengan konsumsi daya yang rendah, *Delay* yang rendah, *throughput* yang tinggi, luas area yang kecil dan kompleksitas yang rendah[3]. Namun untuk mencapai semua persyaratan tersebut hampir tidak memungkinkan, sehingga diperlukan adanya *trade – off* antar parameter performansi jaringan tersebut.

Dua jenis topologi yang paling umum digunakan pada NoC adalah *Mesh* dan *Torus*. Topologi lain yang dapat digunakan dapat berupa *Ring*, *Tree* maupun *Irregular*.

Topologi *mesh* merupakan topologi yang paling sering digunakan pada NoC dikarenakan karakteristiknya yang sederhana dengan skalabilitas tinggi. Pada Topologi 2D mesh, beberapa router dan PE terhubung dengan pola grid yang dikenal sebagai struktur regular. Topologi mesh dapat didefinisikan sebagai *k-ary* n-cubes dimana k adalah jumlah *node* pada setiap dimensinya sedangkan n adalah jumlah dari dimensinya. Misal 2-D mesh memiliki nilai n 2, dan 4-ary 2-cube sama dengan 4x4 mesh atau torus dengan 16 total node. Pada topologi torus, setiap *node* yang berada pada ujung dari jaringan juga saling terhubung sehingga *node* tersebut memiliki jumlah *port* yang sama dengan node yang ada pada tengah jaringan.

Salah satu keuntungan dari topologi mesh adalah mudahnya pendeteksian dan pengisolasian kesalahan pada jaringan. Topologi mesh juga mudah untuk diimplementasikan dan merupakan topologi yang paling sederhana apabila dibandingkan dengan topologi lainnya. Selain itu dengan menggunakan topologi mesh, setiap pesan yang dikirimkan hanya akan mencapai alamat yang dituju.

2.4.2 Teknik Switching

Komponen utama dari arsitektur pada NoC merupakan switch dimana didalam switch data ditransmisikan dari *input port* menuju output port. Dimana data mungkin memerlukan untuk memasuki *buffer* sebelum keluar dari *output port*. Pada switch terdiri dari *arbitration*, *flow control*, algoritma routing yang mengatur kemana dan kapan data akan diteruskan. Arbitration memutuskan kapan data yang datang untuk dilayani oleh router, sedangkan *flow control* menentukan kapan data akan dikirimkan ke router berikutnya dan algoritma *routing* menentukan rute mana yang akan dilalui oleh data tersebut untuk diteruskan selanjutnya melalui *crossbar*[3].

Pada umumnya terdapat dua jenis teknik switching yang sering digunakan yaitu *Store and Forward*, dan *wormhole switching*.

Sedangkan teknik switching yang lainnya dapat berupa *Virtual cut through* maupun circuit switching[5].

Pada *store and forward*, setiap paket yang diterima dirubah menjadi flit. Flit pertama dari paket merupakan *header* flit dan flit terakhir merupakan *tail* flit. *Store and forward* meneruskan header flits ke switch berikutnya hanya jika semua body flits dari paket telah diterima. Sedangkan pada *wormhole switching*, *header* flit dari paket diteruskan ke switch berikutnya sebelumn flit berikutnya dari paket diterima. Sehingga channel *buffer* pada setiap router dapat berukuran sekecil ukuran satu buah flit[5].

2.4.3 Routing Algorithm

Algoritma routing menentukan rute yang akan dilewati oleh pesan yang dikirimkan untuk mencapai tujuan. Tujuan utama dari algoritma routing adalah untuk mendistribusikan trafik dari beberapa node yang berbeda secara merata pada keseluruhan jaringan sehingga dapat menghindari hotspot dan meningkatkan performa jaringan.

Algoritma routing yang digunakan dapat memberikan pengaruh terhadap kompleksitas desain router dan penggunaan area yang kemudian dapat mempengaruhi konsumsi daya pada jaringan secara keseluruhan

Routing dapat diklasifikasn menjadi *source routing* atau *distributed routing* dan *deterministic*, *oblivious* atau *adaptive routing*[2]. Pada *source routing*, node sumber menentukan rute mana yang akan digunakan untuk mengirimkan paket sedangkan pada *distributed routing*, node yang sat tersebut memiliki pesan ke node tujuan pada jaringan akan emnentukan hop / router berikutnya untuk meneruskan paket.

Pada *deterministic routing*, semua paket menggunakan rute yang sama dari node sumber hingga ke node tujuan, sedangkan pada *oblivious routing*, pengambilan keputusan mengenai rute yang dipilih tetap atau tidak berdasarkan kepada kondisi jaringan tetapi tidak mempermasalhkan *congestion* yang terjadi pad jaringan.Pada adative routing, paket akan diteruskan bergantung pada keputusan local dengan mempertimbangkan status dari jaringan dengan tujuan untuk menghindari area yang mengalami *congestion*, dimana pada *adaptive routing*, lebih dari satu rute akan diperhitungkan.

. Salah satu factor yang mempengaruhi performa dari *routing* adalah jumlah dari hop pada rute dan distribusi rute. Selain jenis routing

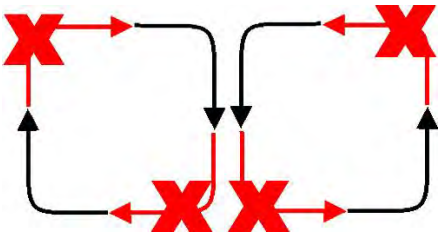
diatas, dengan mengklasifikasn algoritma routing berdasarkan jumlah hop yang digunakan, maka dapat diklasifikasikan menjadi *minimal routing* dan *non minimal routing*. Pada algoritma minimal routing, maka akan menggunakan rute dengan jumlah hop minimal dari sumber ke tujuan. SEdangkan pada non minimal routing, dapat menggunakan rute dengan jumlah hop yang minimum atau tidak.

Algoritma routing yang umum digunakan adalah XY routing yang termasuk kedalam *deterministic routing* dan Odd Even (OE) routing yang termasuk ke dalam *turn based / adaptive routing*.

2.4.3.1 XY Routing Algorithm

Algoritma XY routing merupakan salah satu jenis dari deterministic routing dimana rute yang digunakan merupakan rute yang tetap. XY routing dapat digunakan baik untuk topologi jaringan model biasa maupun yang model tidak biasa. XY routing mengikuti konsep dari minimal turning routing dimana setiap node dinyatakan dengan koordinat (x,y) dari topologi yang digunakan[2].

Pada XY routing, paket yang dikirimkan akan bergerak searah dengan koordinat X hingga menuju kolom node yang dituju. Setelah menemukan kolom dari node tujuan, paket akan bergerak searah koordinat Y hingga mencapai node tujuan. Paket yang dikirimkan tidak dapat bergerak pada arah koordinat Y terlebih dahulu baru kemudian pada arah koordinat X. Sehingga dengan demikian dapat membuat deadlock free. Akan tetapi deterministic routing memiliki potensial untuk terjadinya hotspot jika router tertentu menerima permintaan lebih banyak dari kemampuan router tersebut untuk melayani pada waktu yang bersamaan, sehingga dapat memberikan *delay* yang besar.



Gambar 2.4 Pembatasa Turn pada XY routing

2.5 Flow Control

Flow control merupakan teknik yang digunakan pada jaringan untuk mengontrol proses transmisi data dan untuk menghindari terjadi nya *congestion* pada jaringan dengan trafik yang padat. *Flow control* di desain untuk menghindari penggunaan queue yang berlebihan yang menyebabkan collision pada jaringan.

Pada NoC, *flow control* mengatur pengalokasian dari *buffer* dan link pada setiap router. DImana menentukan kapan *buffer* dan link akan digunakan untuk paket tertentu. *Flow control* juga mengatur pengalokasian dan penggunaan sumber daya yang digunakan bersama – sama diantara semua pesan yang akan dikirimkan[4].

Flow control dapat diklasifikasikan dengan berdasarkan pengalokasian bandwidth kanal nya dan pengalokasian *buffer*. Apabila dilihat dari pengalokasian *buffer* terdapat dua jenis *flow control* yang umum ditemui yaitu *buffer*less *flow control* dan *buffer*ed *flow control*[12].

*Buffer*less *flow control* merupakan teknik *flow control* yang paling sederhana dimana pada *flow control* jenis ini, sumber daya harus segera dialokasikan untuk paket yang datang. Apabila sumber daya yang dibutuhkan tidak tersedia maka paket tersebut akan di drop atau *misroute*. Pada *buffer*less, flit tidak dapat menunggu di router, sehingga *Congestion* di tangani dengan men - drop paket atau melakukan transmisi ulang oleh sumber. Pada jaringan yang menerapkan misroute, maka beberapa paket dari sumber yang sama menuju tujuan yang sama memungkinkan untuk memiliki waktu yang lebih lama dibandingkan paket lainnya. Pada node tujuan, paket yang diterima tidak selalu sesuai urutan, sehingga RNI harus mengurutkan kembali paket yang diterima sesuai urutan sebelum meneruskan ke PE tujuan. Pada jaringan yang menggunakan konsep men – drop paket digunakan sinyal ACK yang dikirimkan oleh node tujuan untuk menandakan bahwa paket yang dikirimkan telah diterima, apabila paket di drop oleh node tujuan maka akan dikirimkan sinyal NACK sehingga node sumber dapat mentransmisikan ulang paket tersebut.

Pada *buffer*ed *flow control* menggunakan *buffer* yang berfungsi untuk menyimpan paket sementara yang diterima oleh router. Paket akan disimpan apabila paket tersebut tidak dapat diteruskan dikarenakan tidak adanya sumber daya yang tersedia. Untuk mengurangi jumlah *buffer* yang digunakan pada router di setiap node beberapa metode digunakan seperti membagi paket menjadi flit. Dalam proses

komunikasi antar dua node, informasi mengenai ketersediaan *buffer* di downstream node harus diketahui oleh upstream node sehingga upstream node dapat memberhentikan transmisi saat slot kosong pada *buffer* tidak tersedia. Proses pemberian informasi atau tanda mengenai kondisi *buffer* disebut juga sebagai backpressure upstream.

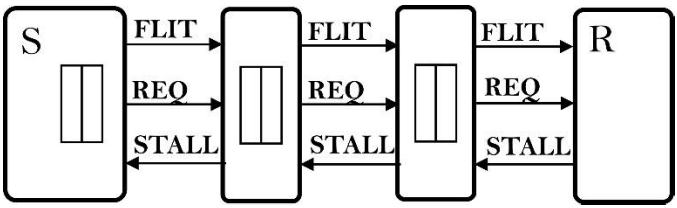
Flow control apabila dilihat dari sisi di tingkatan mana *flow control* tersebut beropreasi dapat pula diklasifikasn kedalam link level *flow control* dan transmission level *flow control*[12].

Pada link level *flow control* merupakan *flow control* jenis node – to – node dimana sebuah node akan mengirimkan informasi mengenai ketersediaan *buffer* nya ke upstream node untuk mencegah terjadinya paket di drop karena *buffer* yang penuh. Link level *flow control* bergantung pada backpressure untuk memberitahu mengenai kondisi *congestion* yang terjadi.

Sedangkan pada transmission level *flow control* disebut sebagai end – to – end *flow control* yang melakukan pengaturan *packet injection rate* pada sumber. Pada transmission level *flow control* terdapat jenis *flow control* berupa credit – based ,window – based dan dedicated signaling *flow control*.

2.5.1 STALL/GO *Flow Control*

STALL/GO *flow control* memerlukan dua buah kabel untuk kontrol, satu untuk flagging data availability yang berarah forward dan yang lainnya untuk signaling kondisi *buffer* baik berupa *buffer* full atau STALL tau *buffer* kosong GO yang berarah backward yang dapat direpresentasikan pada Gambar 2.5[12]. STALL/GO *flow control* dapat diimplementasikan dengan menggunakan FIFO dua level. Pada jaringan yang menerapkan STALL/GO *flow control*, setiap node mengetahui slot kosong pada *buffer* node *downstream*. Dimana saat slot kosong pada buffer node tujuan mencapai batas off, maka sinyal STALL akan di -



Gambar 2.5 Model Jaringan dengan Stall / Go *Flow Control*

kirimkan kembali ke node *upstream* hingga ke node sumber. Kemudian node sumber berhenti mengirimkan flit. Upstream router akan menyimpan flit nya masing – masing dan flit

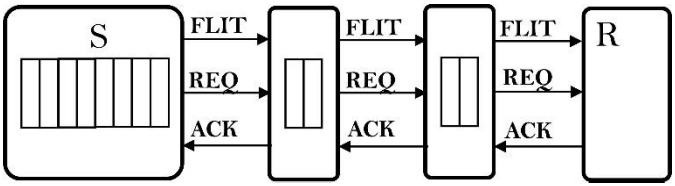
Node tujuan akan memproses data yang ada di *buffer* dan saat slot kosong pada *buffer* mencapai batas *on*, maka node tujuan akan mengirimkan sinyal GO ke node *upstream* hingga ke node sumber sehingga node sumber dapat melanjutkan mengirimkan data sesaat setalah menerima sinyal GO yang dikirimkan oleh node tujuan. Upstream router setelah menerima sinyal GO akan mentransmisikan flit yang berada pada output register pada clock cyle pertama kemudian mentransmisikan flit kedua yang telah disimpan pada *flow control buffer* dan kemudian mengirimkan sinyal GO ke upstream node berikutnya hingga mencapai node sumber[12].

STALL/GO *flow control* memiliki keuntungan berupa round trip *delay* yang rendah dan tidak memerlukan ukuran *buffer* yang besar untuk menyimpan flit.

2.5.2 ACK/NACK *Flow Control*

Pada *flow control* ini, node sumber tidak mengetahui kondisi atau informasi slot kosong pada *buffer* penerima. Node sumber mengirimkan flit secara langsung apabila flit siap. Pada Ack / Nack menggunakan dua kanal forwarding untuk pengiriman flit dan signalling dan satu kanal backward untuk signalling seperti pada gambar Gambar 2.6[5]. Apabila node *downstream* memiliki slot kosong pada *buffer* dan berhasil menerima flit tersebut tanpa *error*, node tersebut akan mengirimkan sinyal ACK, apabila flit tersebut di drop, maka akan mengirimkan sinyal NACK.

Pada node sumber, setelah mentransmisikan flit maka *copyan* dari flit tersebut akan disimpan pada buffer. Apabila node pengirim me -



Gambar 2.6 Model Jaringan dengan Ack /Nack *Flow Control*

nerima ACK , maka *copy* flit yang ada di buffer akan dihapus. Akan tetapi apabila menerima NACK maka akan dilakukan proses transimisi ulang untuk flit yang bersangkutan[12].

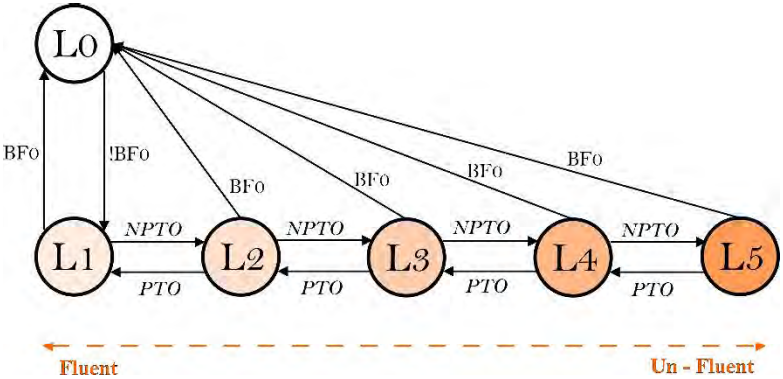
ACK/NACK *flow control* memberikan *throughput* dan *Delay* yang ideal apabila tidak ada NACK yang terjadi, apabila NACK diterima dikarenakan error yang acak, maka pengaruhnya terhadap performansi jaringan dapat diabaikan. Tetapi apabila NACK yang diterima dikarenakan *congestion*, round trip *delay* yang terjadi akan menjadi besar yang dapat menimbulkan penggunaan daya yang semakin besar pula.

2.5.3 Dynamic Multilevel (DyML) *Flow Control*

Konsep dari Dynamic Multilevel *Flow Control* adalah dengan menerapkan konsep in – transit packet untuk meningkatkan *Delay* saat jaringan berada pada beban trafik yang berat dan menggunakan konsep *buffer* fluidity untuk merealisasikan *flow monitor* trafik secara real time. DyML menggunakan parameter berupa *buffer* fluidity untuk memonitor *flow volume* saat terjadi trafik yang padat yang terdeteksi, DYML akan melakukan *buffer*ing dengan konsep in – packet pada wormhole router untuk mencegah terjadinya penggunaan sumberdaya jaringan secara berlebihan. DyML *flow control* memberikan wormhole router kemampuan *flow control* untuk mampu menekan injection paket atau flit ke jaringan saat flow monitor menunjukkan kondisi trafik pada jaringan sedang padat[4].

Dengan menggunakan konsep dasar dari *buffer fluidity*, DyML *Flow Control* menggunakan enam kondisi yang digunakan dalam flow monitor untuk merepresentasikan kondisi jaringan. Kondisi tersebut yaitu L0, L1, L2, L3, L4 dan L5 dimana dapat direpresentasikan seperti pada Gambar 2.7[4].

Dimna L0 hingga L5 merupakan tingkatan dari *buffer* fluidity, BF0 adalah kejadian saat *buffer* fill memiliki nilai sama dengan 0 atau dapat dikatakan sebagai *buffer* dalam keadaan kosong. Sedangkan BF!0 adalah kejadian saat *buffer* fill tidak sama dengan – atau *buffer* dalam keadaan tidak kosong. Sedangkan NPTO adalah Non Pop Time Out dan PTO adalah Pop Time Out. Dimana Pop dapat dikatakan sebagai kejadian dimana *buffer* meneruskan paket ke node berikutnya, sedangkan push didefinisikan sebagai kejadian dimana *buffer* menerima paket dari node lain.



Gambar 2.7 Perpindahan Level pada DyML *Flow Control*

Dengan mengacu pada state diagram diatas, pada DyML *flow control* selain menggunakan *buffer* fill juga menggunakan *buffer* fluidity sebagai parameter yang menjadi pertimbangan. Dimana nilai – nilai *buffer* fluidity dan *buffer* fill untuk setiap kondisi *buffer* dapat dinyatakan sesuai dengan Tabel 2.1 dan Tabel 2.2[4].

Tabel 2.1 Tingkatan *Buffer* Fluidity

Level	***Buffer* Fluidity**
L0	Empty (*buffer* fill = 0)
L1	Fluent
L2	Middle Fluent
L3	Less Fluent
L4	Near Un – Fluent
L5	Un – Fluent

Tabel 2.2 Tingkatan *Buffer* Fill

Level	**Tingkatan *Buffer* Fill**
L0	Full ($buffer$ fill = $buffer$ size)
L1	Near full ($buffer$ fill = $buffer$ size -1)
L2	At least $1/2$ filled ($buffer$ fill $\geq$ $buffer$ size$/2$)
L3	At least $1/4$ filled ($buffer$ fill $\geq$ $buffer$ size$/4$)
L4	At least $1/8$ filled ($buffer$ fill $\geq$ $buffer$ size$/8$)

Kondisi awal dari *buffer* merupakan $L0$ dimana dinyatakan valid saat *buffer* dalam kondisi kosong sedangkah $L1$ hingga $L5$ merpresentasikan lima tingkatan dari *buffer* fluidity dari kondisi fluent hinga un – fluent. Dimana kejadian transisi kondisi didefinisikan dalam NPTO atau Non Pop Time Out yang menyatakan tidak adanya kejadian Pop pada *buffer* selama selang waktu yang telah ditetapkan. Kejadian transisi kondisi berikutnya adalah PTO atau *Pop Time Out* yang merupakan kejadian yang mengindikasikan terdapat beberapa kejadian Pop yang terjadi pada *buffer* saat kondisi saat ini. Apabila *buffer* tidak dapat meneruskan paket ke node berikutnyamk *buffer* akan memiliki kecenderungan untuk menuju kondisi *un – fluent*. Namun sebaliknya apabila jumlah push dan pop yang terjadi pada *buffer* berlangsung secara bergantian, maka *buffer* akan memiliki kemungkinan untuk tetap berada pada kondisi fluent atau $L1$. Dengan demikian, menghitung kejadian push dan pop secara konstan, kondisi trafik yang real time dapat direpresentasikan.

Pada DyML *flow control* dilakukan pendeteksian lokal untuk merefleksikan kondisi trafik jaringan secara keseluruhan. DyML *flow control* tidak mengumpulkan informasi – informasi mengenai kondisi *buffer* dari node – node tetangga untuk dapat melakukan prediksi jaringan atau menggunakan tambahan *buffer* atau kanal untuk dapat memberi tahu node upstream mengenai *congestion.* Tetapi DyML *flow control* bereaksi dan beradaptasi terhadap kondisi trafik jaringa yang berubah – ubah dengan berdasarkan konsep *buffer* fluidity Sehingga keuntungna dari DyML fow control adalah tidak diperlukan adanya proses pertukaran informasi antar node dan tidak memerlukan tambahan bandwidth atau tambahan sinyal control dan tambahan sumberdaya hardware[4].

2.6 Parameter Performansi *Network on Chip*

Tujuan utama dalam mendesain NoC adalah untuk mendapatkan *throughput* yang tinggi, *Delay* yang rendah, penggunaan sumberdaya yang minimal, penggunaan daya yang rendah dan penggunaan luas area yang kecil. Pertimbangan yang paling penting dalam mendesain NoC adalah *trade – off* antara performansi jaringan dengan konsumsi daya.

Pada umumnya, performansi terbagi menjadi dua bagian, performansi komputasi dan performansi komunikasi. Pada NoC, kualitas

dapat dihitung dari segi penggunaan daya, luas area yang digunakan dan performansi jaringan[2].

Beberapa elemen desain yang seperti topologi, algoritma routing dan teknik switching dapat memberikan performansi yang tinggi pada applikasi tertentu namun juga dapat menurunkan performansi pada aplikasi yang lain. Pemilihan implementasi topologi dan protocol komunikasi yang berbeda dapat mempengaruhi parameter performansi. Sehingga dalam mendesain NoC harus memperhatikan pengaruh dari setiap parameter desain terhadap performansi yang dihasilkan.

Delay, *Throughput*, reliability dan penggunaan daya dipengaruhi oleh kapasitas sumber daya, kapasitas kanal dan arsitektur komunikasi. Dimana kapasitas sumberdaya mengindikasikan informasi maksimum yang dapat diproses oleh Processing Element. Dimana merepresentasikan performansi dari router atau processor. Sedangkan kapasitas kanal mengindikasikan jumlah informasi maksimum yang dapat ditransmisikan melalui kanal dengan tingkat reliabilitas tertentu. Semakin tinggi kapasitas dari sumber daya dan jumlah kanal dapat mengurangi *Delay* dan meningkatnak *throughput* dan reliability[15].

2.6.1 Delay

Delay didefinisikan sebagai waktu yang dibutuhkan untuk mentransmisikan paket dari sumber ke tujuan. Pada NoC, *Delay* didefinisikan sebagai waktu antara saat PE sumber mengirimkan bit pertama ke jaringan dan saat PE tujuan menerima bit terakhir dari paket yang dikirimkan[5].

Throughput bergantung pada satuan waktu dikarenakan thorughput dikalkulasi dengan berdasarkan jumlah bit yang diterima per satuan waktu. Pada jaringan yang padat dimana *Delay* meningkat, paket akan berada pada kondisi queue dan meningkatkan waktu kedatangan dari paket, sehingga menurunkan *throughput*. Saat *Delay* meningkat, jaringan tidak dapat menjamin paket yang dikirimkan akan diterima pada tujuan dalam waktu yang diharapkan sehingga mempengaruhi reliability. Peningkatana penggunaan daya berbanding lurus linier terhadap durasi aktif sebuah PE. Sehingga saat *Delay* meningkat, *Delay* akan meningkatkan pula durasi aktif dari PE yang kemudian meningkatkan penggunaan daya[13].

2.6.2 Throughput

Throughput dapat dikatakan sebagai laju data dimana merepresentasikan berapa banyak bit yang diterima pada node tujuan per satuan waktu berupa detik. Pada jaringan, *throughput* bertanggung jawab terhadap laju dimana paket dikirimkan pada jaringan dan merpresentasikan persentase dari total kapasitas jaringan[5].

Pada *network on chip*, *throughput* diukur dengan satuan flits/cycle/IP adapun persamaan yang dapat digunakan yaitu :

$$TP = \frac{TotalFlityangditerima}{Jumlahnode \times Jumlahcycle} \quad (1)$$

Dimana total flit yang diterima merupakan jumlah flit dari semua paket yang sampai pada node tujuan. Jumlah node merupakan jumlah total node pada jaringan dan jumlah cycles merupakan jumlah dari clock cyle yang digunakan antara injeksi paket pertama ke jaringan dan penerimaan paket terakhir pada jaringan. Pada perhitungan *throughput*, hanya data flit yang dipertimbangkan dimana credit flit tidak dipertimbangkan.

2.6.3 Daya

Pada *network on Chip*, daya dihitung dengan satuan joule dimana merupakan daya yang digunakan secara keseluruhan komponen pada jaringan[13]. Dalam suatu jaringan terdapat banyak komponen dan faktor yang dapat mempengaruhi penggunaan daya.

Algoritma routing dan metode selection yang digunakan memiliki nilai penggunaan daya yang berbeda – beda, begitu pula untuk kondisi router yang sedang standby dan aktif memiliki nilai penggunaan daya yang berbeda.

2.6.4 Pengaruh Desain Jaringan terhadap Performansi

Topologi yang berfungsi dalam mendeskripsikan bagaimana sumberdaya saling terhubung pada jaringan merupakan parameter desain yang penting dalam mendesain NoC. Topologi memberikan pengaruh yang besar dalam penggunaan teknik routing dan proses mapping *core* ke node jaringan. Topologi juga memberikan pengaruh terhadap *Delay*, *throughput*, luas area yang digunakan, fault – tolerance dan daya yang digunakan.

Ukuran dari *buffer* pada router memiliki pengaruh pada *Delay* dan *throughput* dimana saat paket diteirma oleh router, maka paket tersebut harus menuju *buffer* pada input port. Saat kapasitas *buffer* di output port melebihi batas maka paket tersebut akan ditunda sehingga meningkatkan *Delay* secara global.

Ukuran dari paket dapat berbeda antara satu dengan lainnya begitu pula dengan format paket tersebut. Perbedaan format dari paket dapat mempengaruhi performansi jaringan terutama *Delay* dan *throughput*. Ukuran paket dinyatakan sebagai ukuran total dari payload dan header. Pada jaringan yang berukuran besar, ukuran paket yang kecil dapat meningkatkan performansi jaringan dikarenakan ukuran paket yang kecil dapat ditransmisikan lebih cepat dibandingkan dengan paket yang berukuran lebih besar.

Dengan meningkatkan injection traffic rate pada jaringan dapat menyebabkan trafik pada jaringan menjadi padat sehingga jaringan dapat terjadi *congestion* yang mempengaruhi performansi jaringan. Akan tetapi apabila kapasita kanal lebih besar dari beban jaringan, maka dengan meningkatkan injection rate akan meningkatkan *throughput*. Akan tetapi apabila kapasitas kanal lebih kecil daripada beban jaringan maka meningkatkan injection rate akan menurunkan *throughput*.

Pada *Network on Chip*, penambahan sumberdaya baru atau komponen baru dapat dikatakan sebagai penambahan kapasitas komunikasi dengan bertambahnya switch dan interkoneksi. Sehingga jaringan harus dapat menjamin pengalokasian bandwidth yang sama rata sesuai dengan tingkat skalabilitas yang dimiliki oleh jaringan tersebut dimana tidak adanya sumberdaya yang memonopoli keseluruhan bandwidth. Penambahan sumber daya baru maka daya yang digunakan juga akan bertambah. Selain itu penambahan komponen hardware antar sumberdaya juga memberikan pengaruh pada luas area yang digunakan bertambah.

Tabel 2.3 menunjukkan parameter – parameter desain yang memberikan pengaruh terhadap performansi jaringan dan Tabel 2.4 menunjukkan parameter – parameter desain yang memberikan pengaruh pada biaya implementasi[13].

Tabel 2.3 Pengaruh Parameter Desain Terhadap Performansi Jaringan

Parameter	Pengaruh pada Performansi Jaringan		
	Delay	***Throughput***	**Reliability**
Topologi	√	√	√
Jum. Sumberdaya	√	√	√
Jenis Aplikasi	√	√	√
Beban Trafik	√	√	√
Injection Rate	√	√	√
Transport Protocol	√	√	√
Ukuran Paket	√	√	√
Algoritma Routing	√	√	×
Ukuran Header	√	√	√
Ukuran Flit	√	√	√
Ukuran *Buffer*	√	√	√
Flow Control	√	√	×

Tabel 2.4 Pengaruh Parameter Desain Terhadap Biaya Implementasi

Parameter	Pengaruh pada Biaya Implementasi		
	Daya	**Luas Area**	**Sumberdaya**
Topologi	√	√	√
Jum. Sumberdaya	√	√	√
Jenis Aplikasi	√	×	√
Beban Trafik	√	×	√
Injection Rate	√	×	√
Transport Protocol	√	×	√
Ukuran Paket	√	×	×
Algoritma Routing	√	×	√
Ukuran Header	√	×	×
Ukuran Flit	√	×	×
Ukuran *Buffer*	√	√	×
Flow Control	√	×	√

BAB 3
PERANCANGAN DAN IMPLEMENTASI *FLOW CONTROL*

Pada bab ini dibahas mengenai metode yang digunakan untuk melakukan pengujian terhadap metode *flow control* pada NoC. Tujuan yang ingin dicapai adalah mendapatkan hasil parameter unjuk kerja berupa *throughput* dan titik saturasi dari metode – metode STALL/GO *flow control*, ACK/NACK *flow control* dan DyML *flow control* pada kondisi jaringan yang tersaturasi. Dimana parameter desain yang digunakan merupakan kombinasi dari ukuran *buffer*, ukuran paket dan *packet injection rate*.

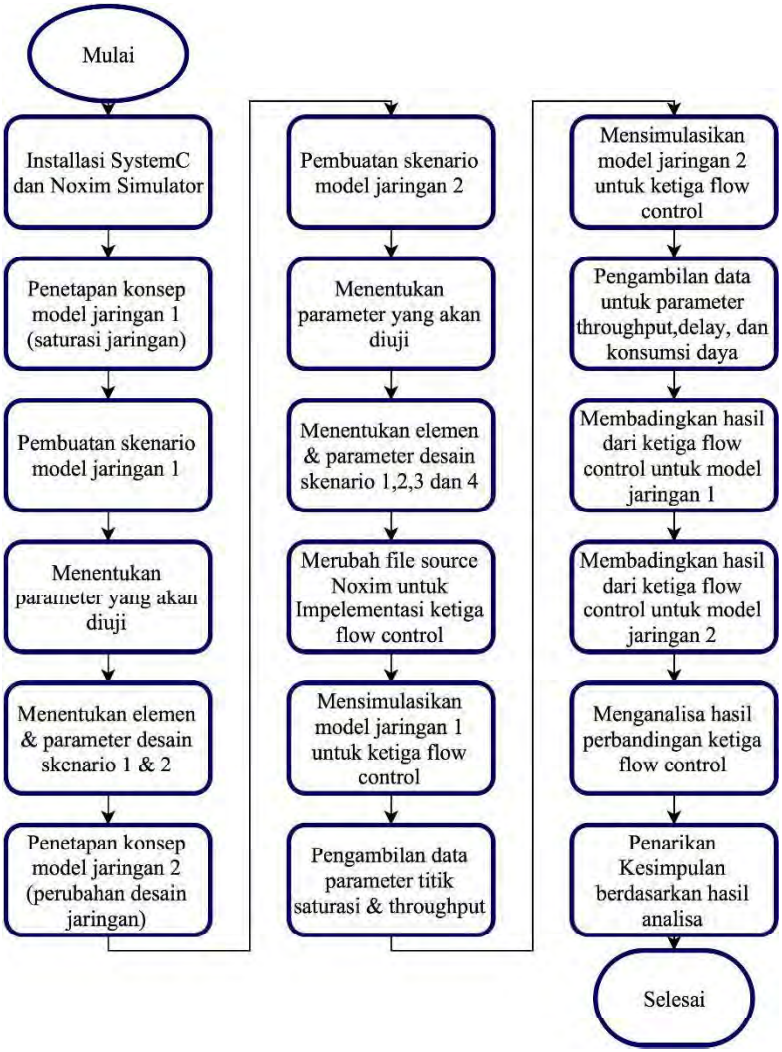
Selain itu juga bertujuan untuk mendapatkan pengaruh dari STALL/GO *flow control*, ACK/NACK *flow control* dan DyML f*low control* terhadap performansi jaringan berupa *throughput*, *Delay*, dan daya yang digunakan. Dengan parameter desain yang digunakan berupa ukuran jaringan, *packet injection rate*, algoritma routing, ukuran paket dan ukuran *buffer*.

Dalam bab ini juga dibahas mengenai hal- hal yang berkaitan dengan perancangan desain jaringan yang digunakan dalam tugas akhir ini beserta langkah – langkah yang dilakukan untuk mendapatkan parameter – parameter desain yang ditentukan. Adapun alur dari penelitian Tugas Akhir ini dapat direpresentasikan dalam Gambar 3.1

3.1 Software Pendukung

Software pendukung yang digunakan untuk melakukan pengimplementasian flow conrol secara simulasi terhadap parameter desain dan desain jaringan yang telah ditentukan untuk menguji parameter performansi yang diinginkan adalah Noxim Simulator.

Noxim merupakan simulator untuk *Network on Chip* yang dikembangkan dengan menggunakan SystemC dibawah lisensi GPL. Dalam kaitannya dengan parameter desain jaringan, Noxim menyediakan beberapa parameter yang dapat diubah – ubah yaitu ukuran paket dan *packet injection rate*. Noxim juga menyediakan beberapa pilihan model trafik yang dapat digunakan yaitu random, transpose1, transpose2, bit reversal, butterfly dan shuffle. Sedangkan



Gambar 3.1 Alur Diagram Pengerjaan Tugas Akhir

mengenai parameter performansi pada NoC yang dapat diberikan oleh Noxim adalah *throughput, Delay* dan penggunaan daya rata – rata

3.1.1 Proses Instalasi

Noxim ditulis dengan menggunakan bahasa C++ dan systemC library sehingga Noxim dapat digunakan di berbagai platform yang mendukung systemC. Pada umumnya Noxim dioperasikan dengan operating system GNU/Linux akan tetapi Noxim juga dapat beroperasi dengan operating system Apple Mac OS Xdan Sun Solaris dengan GCC begitu pula untuk Microsoft Windows dengan menggunakan Visual C++ atau dengan menggunakan Cygwin dengan GCC.

Sebelum menginstall Noxim harus dilakukan instalasi systemC terlebih dahulu. Untuk dapat mengcompile systemC diperlukan C++ compiler. Untuk operating system Linux, dapat menggunakan langkah – langkah berikut untuk menginstall systemC

1. Buka terminal dan masukkan perintah

```
sudo apt-get install build-essential
```

2. Setelah proses download selesai, untar file dengan perintah

```
mv systemc-2.3.1.tgz systemc-2.3.1.tar
tar xvf sysemc-2.3.1.tar
```

3. Pada terminal Masuk ke directory yang telah terbuat dan masukkan perintah

```
cd downloads/systemc-2.3.1
mkdir objdir
cd objdir
export CXX=g++
../configure
make
make install
```

4. Masuk ke directory Noxim > bin
5. Edit file Makefile.defs
6. tambahkan systemC installation path pada file tersebut
7. Pada terminal masukkan perintah

```
cd Noxim/bin
make
```

3.1.2 Proses Simulasi

Noxim merupakan simulator yang memiliki interface berupa command line. Adapun perintah – perintah dasar yang dapat digunakan yaitu :

```
Noxim [option]
```

Untuk dapat mengetahui option yang dapat digunakan dan kegunaan masing – masing option yang dapat digunakan secara default dapat menggunakan perintah

```
./Noxim – help
```

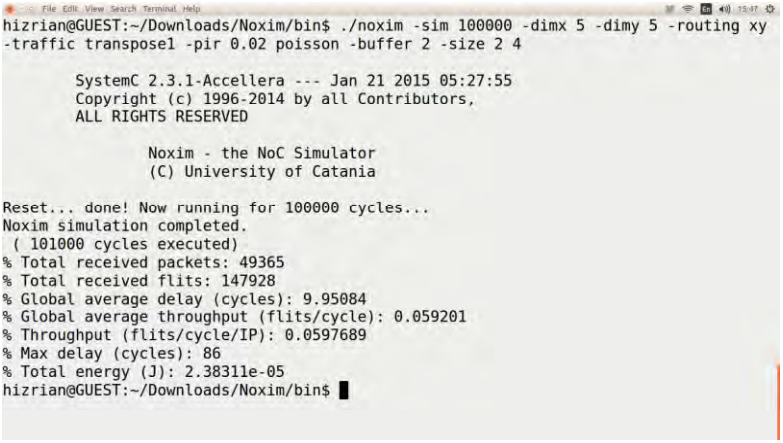
Dimana option yang dapat dilakukan adalah :

- verbose *N*
- trace *FILENAME*
- dimx *N*
- dimy *N*
- *buffer N*
- routing *TYPE*
- sel *TYPE*
- pir *R TYPE*
- traffic *TYPE*
- hs *ID P*
- warmup *N*
- seed *N*
- detailed
- volume *N*
- sim *N*

Gambar 3.2 menunjukkan tampilan dasar dari Noxim yang mensimulasikan jaringan berukuran 5 x 5 dengan simulation time selama 100000 cycle dan routing xy dengan traffic transpose1

3.2 Desain Jaringan

Pada tahapan desain jaringan dilakukan dua tahap pendesainan untuk dua kondisi jaringan yang berbeda. Dimana yang pertama adalah desain



Gambar 3.2 Contoh Simulasi Jaringan dengan Noxim

jaringan untuk menganalisis pengaruh *flow control* pada kondisi jaringan yang mengalami saturasi dan yang kedua adalah desain jaringan untuk menganalisis pengaruh flow conrol dan perubahan parameter desain terhadap parameter – parameter performansi jaringan.

3.2.1 Desain Jaringan Kondisi Saturasi

Dalam mendesain kondisi jaringan yang tersaturasi, langkah pertama yang dilakukan adalah dengan menentukan skenario dari kombinasi parameter desain yang ada untuk mencapai kondisi saturasi. Berikutnya menentukan range dari parameter desain tersebut dengan tujuan untuk menentukan titik saturasi dan nilai thorughput maksimum yang diberikan oleh masing – masing parameter desain yang digunakan serta untuk dapat menentukan pengaruh dari *flow control* terhadap titik saturasi dan nilai *throughput* maksimum.

Beberapa parameter desain yang mempengaruhi perubahan nilai dari *throughput* diantaranya adalah *Packet injection rate*, Transport Protocol, Ukuran Paket, Algoritma Routing, Ukuran flit, Ukuran Bufer dan *Flow Control*.

Dari parameter desain yang mempengaruhi nilai dari *throughput*, beberapa parameter desain akan menyebabkan jaringan menjadi saturasi. Kombinasi dari parameter desain tersebut dijadikan

skenario untuk memperoleh hasil pengaruh *flow control* terhadap jaringan yang tersaturasi. Adapun skenario yang digunakan yaitu :

1. Kombinasi Ukuran *Buffer* dan *Packet injection rate*
2. Kombinasi *Packet injection rate* dan Ukuran Paket

Dengan menggunakan Noxim simulator, maka elemen dan parameter desain yang digunakan beserta batas nilainya sesuai dengan Tabel 3.1,Tabel 3.2, dan Tabel 3.3.

Tabel 3.1 Elemen Desain Model Jaringan 1

No	Elemen Desain	Nilai
1	Simulation Time	100000 cycles
2	Topologi	2D - Mesh
3	Ukuran Jaringan	5 x 5
4	Algoritma Routing	XY routing
5	Selection Function	Random
6	Model Trafik	Transpose1

Tabel 3.2 Parameter Desain Model Jaringan 1 Skenario 1

No	Elemen Desain	Nilai
1	Ukuran *Buffer*	2 – 10 flits
2	*Packet injection rate*	0.02 – 0.3
3	Ukuran Paket	2 – 4 flits
4	*Flow Control*	Default STALL/GO ACK/NACK DyML

Tabel 3.3 Parameter Desain Model Jaringan 1 Skenario 2

No	Parameter Desain	Nilai
1	Ukuran Paket	2 – 22 flits
2	*Packet injection rate*	0.04 – 0.08
3	Ukuran *Buffer*	8 flits
4	*Flow Control*	Default *Flow Control* (NFC) STALL/GO (SGFC) ACK/NACK (ACFC) DyML (DyML)

3.2.2 Desain Jaringan dengan Parameter Desain berubah - ubah

Model jaringan 2 bertujuan untuk mengamati pengaruh dari parameter desain terhadap perubahan parameter performansi serta pengaruh dari penggunaan *flow control* terhadap perubahan parameter performansi. Adapun parameter perfomnsi yang akan diamati berupa *throughput*, *Delay*, reliability, jumlah flit yang diterima, jumlah paket yang diterima dan daya yang digunakan.

Model jaringan 2 yang digunakan dibagi menjadi kedalam empat skenario sesuai dengan parameter desain yang akan diamati pengaruhnya terhadap parameter performansi yaitu.

1. Skenario 1 Perubahan ukuran jaringan
2. Skenario 2 Perubahan *Packet injection rate*
3. Skenario 3 Perubahan ukuran paket
4. Skenario 4 Perubahan ukuran buffer

Adapun nilai dari elemen dan parameter desain untuk keempat skenario sesuai dengan Tabel 3.4, Tabel 3.5,
Tabel **3.6**, Tabel 3.7, dan Tabel 3.8.

Tabel 3.4 Elemen Desain Model Jaringan 2

No	Elemen Desain	Nilai
1	Simulation Time	100000 cycles
2	Topologi	2D - Mesh
3	Algoritma Routing	XY routing
4	Selection Function	Random
5	Model Trafik	Transpose1

Tabel 3.5 Parameter Desain Model Jaringan 2 Skenario 1

Skenario 1 – Ukuran Jaringan		
No	**Parameter Desain**	**Nilai**
1	*Packet injection rate*	0.06
2	Ukuran Paket	2 - 4 flit
3	Ukuran *Buffer*	8 flit
4	*Flow Control*	Default STALL/GO ACK/NACK DyML
5	Ukuran Jaringan	2 x 2 – 9 x 9

Tabel 3.6 Parameter Desain Model Jaringan 2 Skenario 2

Skenario 2 – *Packet injection rate*		
No	**Parameter Desain**	**Nilai**
1	*Packet injection rate*	0.02 – 0.18
2	Ukuran Paket	2 – 4 flit
3	Ukuran *Buffer*	8 flit
4	*Flow Control*	Default STALL/GO ACK/NACK DyML
5	Ukuran Jaringan	5 x 5

Tabel 3.7 Parameter Desain Model Jaringan 2 Skenario 3

Skenario 3 – Ukuran Paket		
No	**Parameter Desain**	**Nilai**
1	*Packet injection rate*	0.06
2	Ukuran Paket	2 – 18 flit
3	Ukuran *Buffer*	8 flit
4	*Flow Control*	Default STALL/GO ACK/NACK DyML
5	Ukuran Jaringan	5 x 5

Tabel 3.8 Parameter Desain Model Jaringan 2 Skenario 4

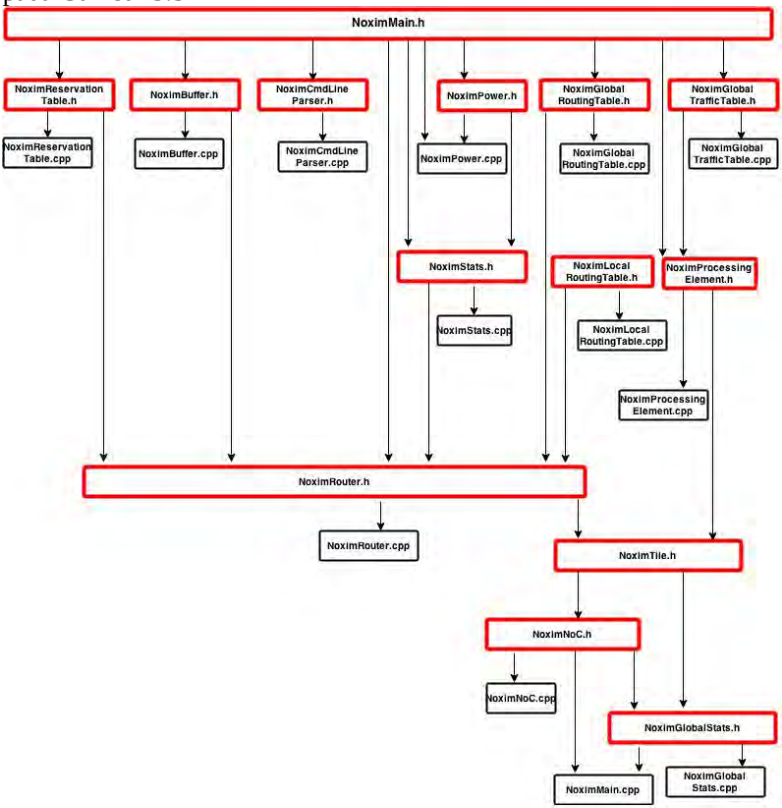
Skenario 4 – Ukuran *Buffer*		
No	**Parameter Desain**	**Nilai**
1	*Packet injection rate*	0.06
2	Ukuran Paket	2 4 flit
3	Ukuran *Buffer*	1 – 17 flit
4	*Flow Control*	Default STALL/GO ACK/NACK DyML
5	Ukuran Jaringan	5 x 5

3.3 Implementasi *Flow Control*

Setelah didapat algoritma untuk masing – masing metode *flow control* maka proses berikutnya adalah mengimplementasikan ke dalam desain jaringan yang telah dibuat.

Karena Noxim merupakan simulator yang menggunakan systemC, maka algoritma algoritma tersebut harus diubah ke dalam bentuk bahasa C++ terlebih dahulu.

Untuk dapat mengimplementasikan *flow control* pada Noxim, maka diperlukan perubahan pada file source pada Noxim simulator. Dimana struktur file source dari Noxim Simulator itu sendiri seperti pada Gambar 3.3



Gambar 3.3 Struktur File Konfigurasi pada Noxim

Sehingga file - file yang dirubah untuk penerapan *flow control* adalah seperti berikut

- NoximProcessingElement untuk mengatur proses transmisi dan proses penerimaan di ProcessingElement tiap router :
- NoximRouter digunakan untuk mengatur proses – proses yang dilakukan pada router seperti proses pengiriman dan penerimaan flit / paket dan proses routing.
- NoximTile dan NoximNoC untuk Pengaturan kanal forward dan kanal backward
- NoximMain dapat dilakukan pendefinsian parameter – parameter global yang dapat digunakan.
- Noxim*Buffer* digunakan untuk mengatur bagaimana *buffer* beroperasi

3.4 Pengambilan Data

Pada proses pengambila data, parameter performansi jaringan pada kedua desain yang telah dibuat akan diperoleh. Dimana dalam proses pengambilan data dilakukan secara bergantian untuk masing – masing desain jaringan.

3.4.1 Kondisi Saturasi

Seperti yang telah dijelaskan, parameter performansi yang akan diamati adalah titik saturasi dan nilai *throughput* makimsum. Berikut ini adalah contoh perintah – perintah yang digunakan pada Noxim untuk masing – masing *flow control* dan untuk masing – masing skenario yang telah ditetapkan

Adapun contoh perintah yang digunakan adalah sebagai berikut

Skenario 1

```
./Noxim -sim 100000 -dimx 5 -dimy 5 -routing xy -traffic transpose1 -PIR 0.02 poisson -buffer 2 -size 2 4
```

Skenario 2

```
./Noxim -sim 100000 -dimx 5 -dimy 5 -routing xy -traffic transpose1 -PIR 0.04 poisson  -buffer 8 -size 2 4
```

Untuk pengambilan data tanpa *flow control* maupun dengan menggunakan ketiga metode *flow control* tidak terdapat perbedaaan perintah. Hal ini dikarenakan implementasi *flow control* sudah secara

langsung diterapkan saat merubah file source pada Noxim yaitu NoximRouter.h, NoximRouter.cpp, NoximProcessingElement.h, NoximProcessingElement.cpp, NoximNoC.h, NoximNoC.cpp, NoximTile.h

Gambar 3.4 adalah contoh hasil pengambilan data untuk skenario 1 dan Gambar 3.5. adalah contoh hasil pengambilan data untuk skenario 2 sebagai berikut

```
hizrian@GUEST:~/Downloads/Noxim/bin$ ./noxim -sim 100000 -dimx 5 -dimy 5 -routing xy
-traffic transpose1 -pir 0.02 poisson -buffer 2 -size 2 4

        SystemC 2.3.1-Accellera --- Jan 21 2015 05:27:55
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED

                Noxim - the NoC Simulator
                (C) University of Catania

Reset... done! Now running for 100000 cycles...
Noxim simulation completed.
 ( 101000 cycles executed)
% Total received packets: 49365
% Total received flits: 147928
% Global average delay (cycles): 9.95084
% Global average throughput (flits/cycle): 0.059201
% Throughput (flits/cycle/IP): 0.0597689
% Max delay (cycles): 86
% Total energy (J): 2.38311e-05
hizrian@GUEST:~/Downloads/Noxim/bin$
```

Gambar 3.4 Contoh Pengambilan Data untuk Model 1 Skenario 1

```
hizrian@GUEST:~/Downloads/Noxim/bin$ ./noxim -sim 100000 -dimx 5 -dimy 5 -routing xy
-traffic transpose1 -pir 0.04 poisson -buffer 8 -size 2 4

        SystemC 2.3.1-Accellera --- Jan 21 2015 05:27:55
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED

                Noxim - the NoC Simulator
                (C) University of Catania

Reset... done! Now running for 100000 cycles...
Noxim simulation completed.
 ( 101000 cycles executed)
% Total received packets: 98830
% Total received flits: 296507
% Global average delay (cycles): 31.2364
% Global average throughput (flits/cycle): 0.118627
% Throughput (flits/cycle/IP): 0.119801
% Max delay (cycles): 1097
% Total energy (J): 3.91244e-05
hizrian@GUEST:~/Downloads/Noxim/bin$
```

Gambar 3.5 Contoh Pengambilan Data untuk Model 1 Skenario 2

3.4.2 Pengaruh Parameter Desain terhadap Performansi

Parameter performansi yang akan diambil adalah *throughput*, *Delay*, reliability dan daya yang digunakan. Dimana parameter desain yang digunakan sesuai dengan yang desain jaringan yang telah dibuat.

Adapun contoh perintah yang digunakan untuk masing – masing model adalah sebagai berikut :

Skenario 1 – Ukuran Jaringan

```
./Noxim -sim 100000 -routing xy -traffic transpose1 -PIR 0.06 poisson -size 2 4 -buffer 8 -dimx 2 -dimy 2
```

```
./Noxim -sim 100000 -routing xy -traffic transpose1 -PIR 0.06 poisson -size 2 4 -buffer 8 -dimx 4 -dimy 4
```

Skenario 2 – *Packet injection rate*

```
./Noxim -sim 100000 -routing xy -traffic transpose1 -PIR 0.06 poisson -size 2 4 -buffer 8 -dimx 5 -dimy 5
```

```
./Noxim -sim 100000 -routing xy -traffic transpose1 -PIR 0.08 poisson -size 2 4 -buffer 8 -dimx 5 -dimy 5
```

Skenario 3 – Ukuran Paket

```
./Noxim -sim 100000 -routing xy -traffic transpose1 -PIR 0.06 poisson -size 2 4 -buffer 8 -dimx 5 -dimy 5
```

```
./Noxim -sim 100000 -routing xy -traffic transpose1 -PIR 0.06 poisson -size 4 6 -buffer 8 -dimx 5 -dimy 5
```

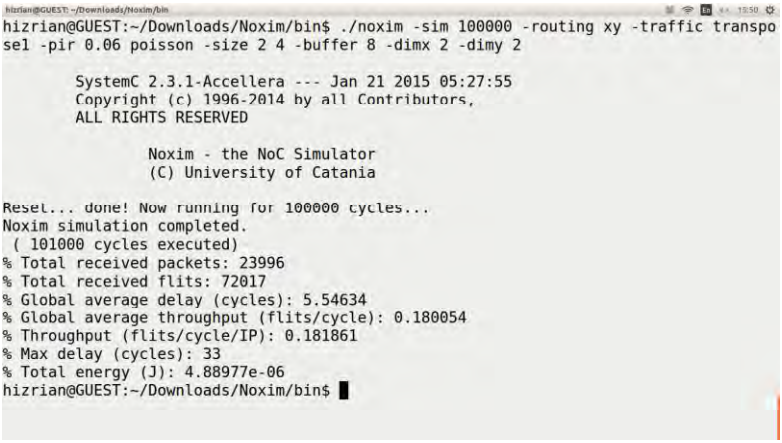
Skenario 4 – Ukuran *Buffer*

```
./Noxim -sim 100000 -routing xy -traffic transpose1 -PIR 0.06 poisson -size 2 4 -buffer 1 -dimx 5 -dimy 5
```

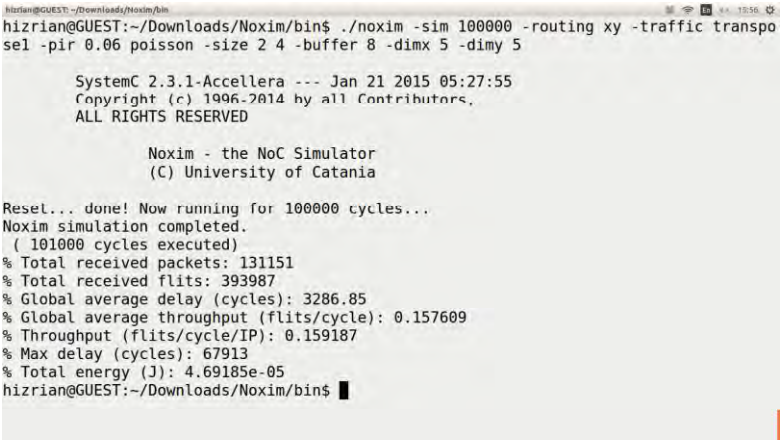
```
./Noxim -sim 100000 -routing xy -traffic transpose1 -PIR 0.06 poisson -size 2 4 -buffer 2 -dimx 5 -dimy 5
```

Gambar 3.6 adalah contoh hasil yang diperoleh untuk simulasi dengan ukuran jaringan yang berubah – ubah. Gambar 3.7 adalah contoh

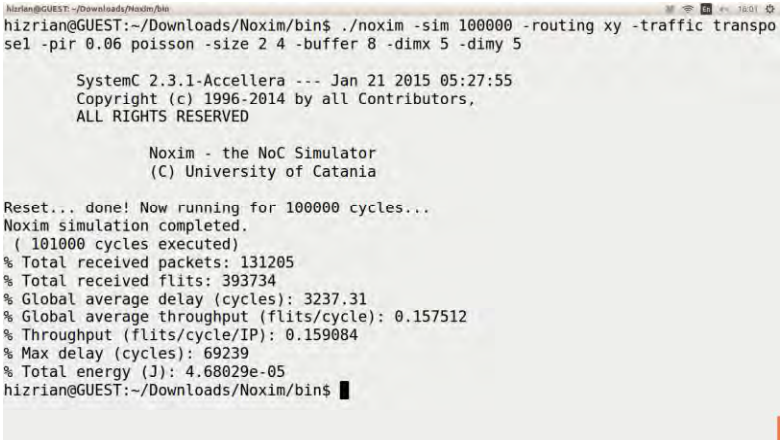
hasil yang diperoleh untuk simulasi dengan *Packet injection rate* yang berubah – ubah. Gambar 3.8 adalah contoh hasil yang diperoleh untuk simulasi dengan ukuran paket yang berubah – ubah. Gambar 3.9 adalah contoh hasil yang diperoleh untuk simulasi dengan ukuran buffer yang berubah – ubah.



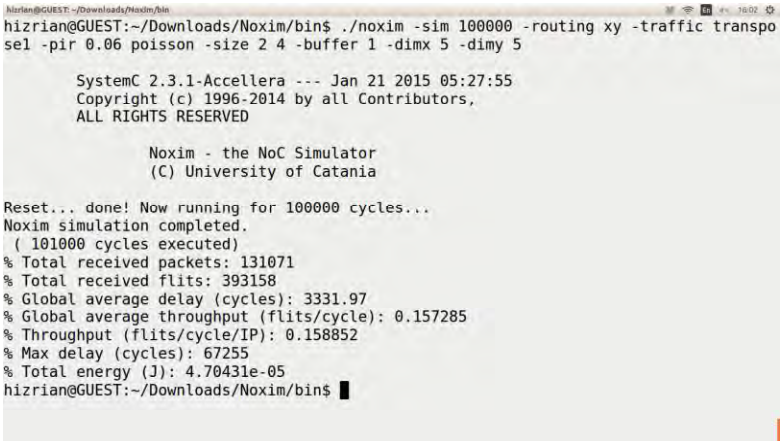
Gambar 3.6 Contoh Pengambilan Data Model 2 Skenario 1



Gambar 3.7 Contoh Pengambilan Data Model 2 Skenario 2



Gambar 3.8 Contoh Pengambilan Data Model 2 Skenario 3



Gambar 3.9 Contoh Pengambilan Data Model 2 Skenario 4

Untuk pengambilan data tanpa *flow control* maupun dengan menggunakan ketiga metode *flow control* tidak terdapat perbedaaan perintah. Hal ini dikarenakan implementasi *flow control* sudah secara langsung diterapkan saat merubah file *source* pada Noxim yaitu NoximRouter.h, NoximRouter.cpp, NoximProcessingElement.h, NoximProcessingElement.cpp, NoximNoC.h, NoximNoC.cpp, NoximTile.h

BAB 4
ANALISIS DATA DAN PEMBAHASAN

Pada bab ini ditampilkan data yang telah diperoleh melalui pengujian tiga metode *flow control* dalam beberapa kondisi jaringan yang telah didesain. Dimana ketiga *flow control* tersebut yaitu STALL/GO *flow control*, ACK/NACK *flow control* dan Dynamic MultiLevel (DyML) *Flow Control*. Kemudian dari hasil pengujian tersebut dapat dilakukan analisis pengaruh penerapan *flow control* terhadap parameter – parameter performansi jaringan.

Analisis performansi dilakukan dengan cara membandingkan parameter – parameter performansi yang diperoleh dari jaringan yang menggunakan *flow control* dan dengan jaringan yang menggunakan teknik *flow control* dasar / *defaut*. Kemudian dilakukan pula perbandingan terhadap parameter performansi jaringan pada ketiga jaringan dengan menggunakan metode *flow control* yang berbeda – beda untuk diperoleh metode *flow control* yang memberikan hasil terbaik dalam meningkatkan performansi jaringan untuk model jaringan tertentu.

Kemudian dari data – data yang diperoleh tersebut ditampilkan baik dalam bentuk grafik maupun tabel dan digunakan sebagai dasar untuk penarikan kesimpulan.

4.1 Analisis Kondisi Saturasi

Setiap jaringan komunikasi memiliki batas maksimum performansi yang dapat disebut sebagai bandwidth limit. Pada sub bab ini, akan dianalisis pengaruh dari kombinasi penggunaan parameter desain antara ukuran *buffer* dan *Packet injection rate*, dan antara *Packet injection rate* dan Ukuran Paket. Parameter yang akan diperoleh yaitu dimana jaringan akan tersaturasi dan berapa nilai *throughput* maksimum yang dicapai oleh masing – masing kombinasi tersebut.

Salah satu parameter yang mempengaruhi nilai *throughput* dari suatu jaringan adalah *flow control*. Dengan menggunakan *flow control* akan dapat mengatasi *congestion* pada router, sehingga akan meningkatkan *throughput*. Oleh karena itu, dilakukan analisis pengaruh dari penggunaan tiga metode *flow control* yang berbeda – beda terhadap *throughput* maksimum dan titik saturasi pada jaringan.

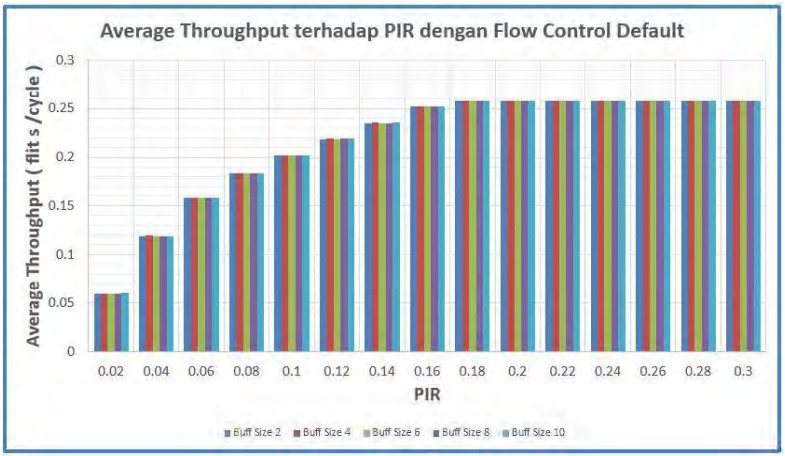
4.1.1 Kombinasi *Buffer* Size dan *Packet injection rate*

Beberapa parameter yang mempengaruhi nilai dari *throughput* adalah ukuran *buffer* dan *packet injection rate*. Dengan melakukan perubahan nilai dari *packet injection rate* dengan ukuran *buffer* tertentu akan diperoleh nilai *throughput* yang berubah – ubah.

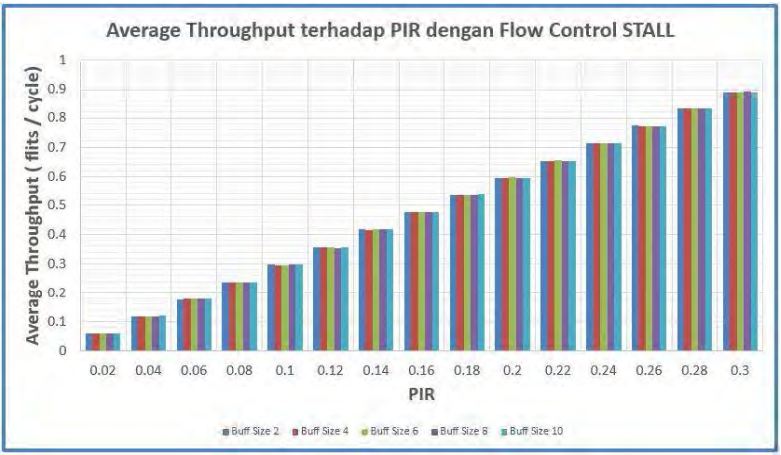
Gambar **4.1** menunjukkan hasil pengambilan data untuk kombinasi ukuran *buffer* dan *packet injection rate* dengan *flow control* default

Untuk jaringan yang tanpa menggunakan *flow control*, saat nilai dari *packet injection rate* diperbesar diatas 0.18 menyebabkan jaringan mengalami saturasi. Hal tersebut dikarenakan apabila *packet injection rate* diperbesar maka jumlah flit yang ada pada jaringan akan semakin banyak. Pada jaringan tanpa *flow control* , flit dikirimkan dengan sliding window bernilai 1, sehingga dengan ukuran buffer yang tetap apabila jumlah flit yang dikirimkan meningkat akan menyebabkan terjadinya saturasi dimana peningkatan jumlah flit tidak lagi mempengaruhi jumlah flit yang diterima.

Gambar 4.2 menunjukkan hasil pengambilan data untuk kombinasi ukuran *buffer* dan *packet injection rate* dengan *flow control* Stall / Go



Gambar 4.1 Hasil Skenario 1 dengan *Flow Control* Default



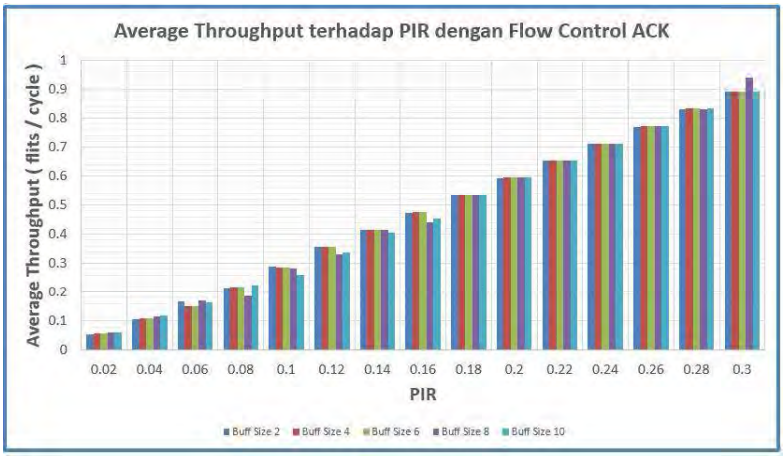
Gambar 4.2 Hasil Skenario 1 dengan *Flow Control* STALL

Jaringan yang menerapkan *flow control* Stall saat nilai *packet injection rate* diperbesar tidak menyebabkan terjadinya saturasi. Hal tersebut dikarenakan jaringan dapat terus mengirimkan flit selama signaling bertanda GO. Saat *packet injection rate* diperbesar yang mengakibatkan jumlah flit yang dikirimkan meningkat, dengan ukuran buffer yang tetap maka peruahan signalling dari STALL ke GO dipengaruhi oleh batas Off dan batas On yang bergantung pada ukuran buffer. Dengan demikian meskipun *packet injection rate* diperbesar, pengirim dapat terus mengirimkan flit.

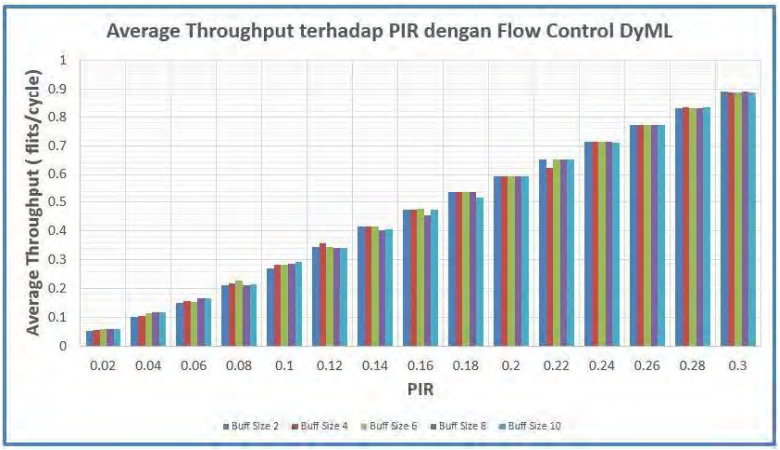
Gambar 4.3 menunjukkan hasil pengambilan data untuk kombinasi ukuran *buffer* dan *packet injection rate* dengan *flow control* Ack / Nack

Jaringan yang menggunakan flow control Ack/Nack tidak menyebabkan saturasi pada jaringan meskipun nilai dari *packet injection rate* diperbesar. Hal tersebut dikarenakan pengirim dapat terus mengirimkan paket sebanyak - banyaknya selama menerima Ack , dan meskipun terjadi paket drop dan pengirim menerima Nack , pengirim tetap dapat mengirimkan paket sebanyak – banyaknya meskipun dilakukan proses transmisi ulang.

Gambar 4.4 menunjukkan hasil pengambilan data untuk kombinasi ukuran *buffer* dan *packet injection rate* dengan *flow control* Dynamic Multi Level



Gambar 4.3 Hasil Skenario 1 dengan *Flow Control* ACK



Gambar 4.4 Hasil Skenario 1 dengan *Flow Control* DyML

Dari data yang diperoleh dapat terlihat bahwa untuk semua ukuran *buffer* yang dicoba dengan menggunakan Dynamic Multi Level *flow control*, *throughput* jaringan terus meningkat seiring dengan meningkatnya *packet injection rate*. Hal ini dikarenakan pada *flow control* DyML flit dapat dikirimkan secara terus menerus selama signaling bertanda 'GO'. Perubahan signalling menuju 'STALL'

dipengaruhi oleh batas off dan perubahan menuju 'GO' dipengaruhi oleh batas on dimana batas off dan batas on nilainya berubah – ubah sesuai dengan ukuran *buffer*. Selain itu *buffer* fluidity juga mempengaruhi kemungkinan terjadinya 'STALL'. Sehingga dengan meningkatkan *packet injection rate* yang berarti menambah jumlah flit yang dikirimkan flit dapat selalu diterima oleh penerima.

Tabel 4.1 menunjukkan perbandingan kombinasi desain parameter *packet injection rate* dengan ukuran *buffer* yang berbeda - beda untuk ketiga metode *flow control* yang digunakan dengan parameter yang diamati merupakan titik saturasi dan *throughput* maksimum

Tabel 4.1 Perbandingan *Throughput* Maksimum dan Titik Saturasi Skenario 1

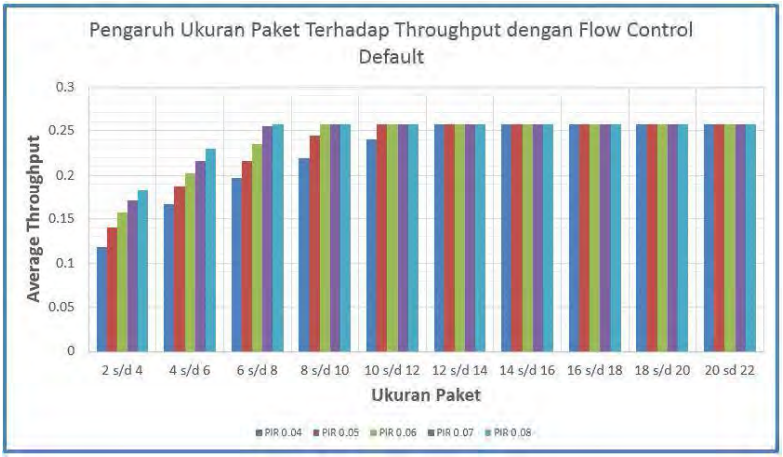
No	*Flow Control*	*Throughput* Maksimum	Titik Saturasi (PIR)
1	Default	0.25742 flits / Cycle	0.18
2	Stall / Go	Tidak Tersaturasi	Tidak Tersaturasi
3	Ack / Nack	Tidak Tersaturasi	Tidak Tersaturasi
4	Dynamic Multi Level	Tidak Tersaturasi	Tidak Tersaturasi

4.1.2 Kombinasi *Packet injection rate* dan Packet Size

Selain itu parameter yang mempengaruhi nilai dari *throughput* selain *packet injection rate* adalah ukuran dari paket. Dengan melakukan perubahan nilai dari ukuran paket dengan *packet injection rate* tertentu akan diperoleh nilai *throughput* yang berubah – ubah. Dimana dengan menggunakan tiga metode *flow control* yang berbeda akan memberikan hasil *throughput* yang berbeda beda.

Gambar 4.5 menunjukkan hasil pengambilan data untuk kombinasi ukuran paket dan *packet injection rate* dengan *flow control* default

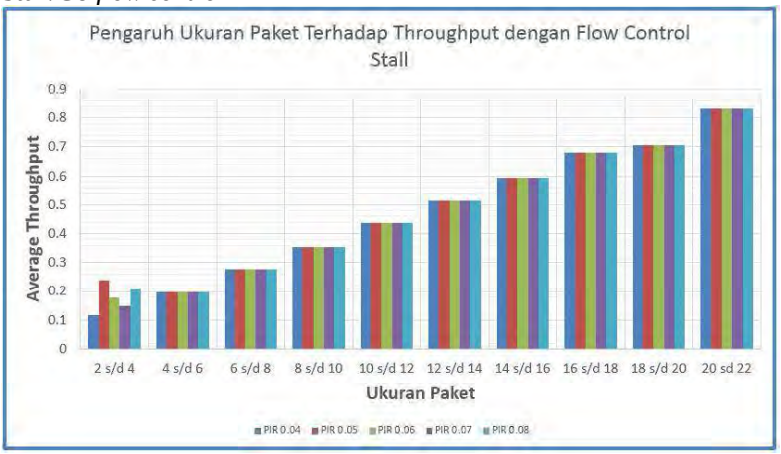
Dari data yang diperoleh dapat terlihat bahwa saat nilai ukuran paket 10 – 12 flit, untuk semua *packet injection rate* yang dicoba, *throughput* jaringan mulai tersaturasi. Hal ini dikarenakan penggunaan sliding window 1 untuk mengirimkan flit dan ukuran dari buffer yang tetap nilainya. Sehingga meskipun ukuran paket diperbesar, jumlah flit



Gambar 4.5 Hasil Skenario 2 dengan *Flow Control* Default

yang dapat diterima tidak akan bertambah banyak dan penambahan flit pada jaringan tidak akan mempengaruhi flit drop maupun Congestion lagi sehingga terjadi saturasi.

Gambar 4.6 menunjukkan hasil pengambilan data untuk kombinasi ukuran paket dan *packet injection rate* dengan *flow control* Stall/Go *flow control*

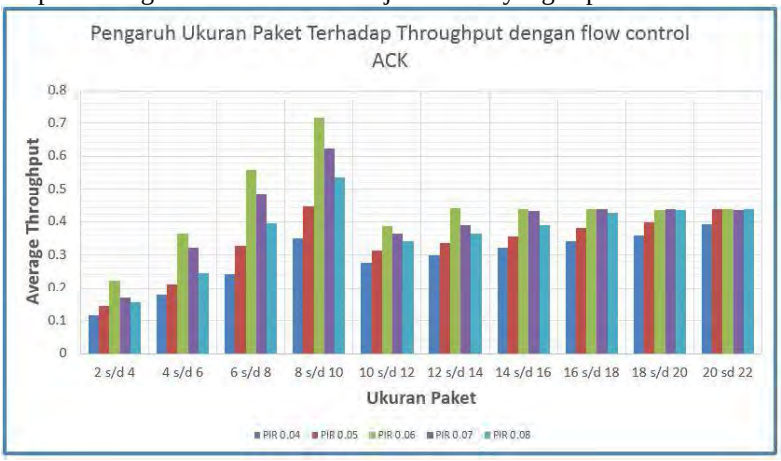


Gambar 4.6 Hasil Skenario 2 dengan *Flow Control* Stall

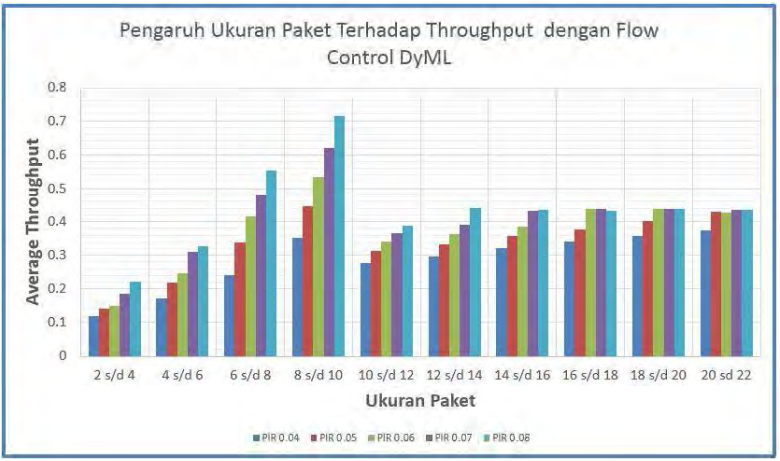
Sedangkan untuk *flow control* Stall, belum terjadi saturasi seiring dengan peningkatan ukuran paket hingga 20 – 22 flit. Hal tersebut dikarenakan flit dapat dikirimkan terus menerus selama signalling bertanda 'GO' dan perubahan signalling menjadi 'STALL' dipengaruhi oleh ukuran *buffer* yang mempenaruhi batas off dan batas on. Sehingga selama signalling bertanda 'GO' flit dapat terus dikirimkan dan diterima yang mana menyebabkan kondisi saturasi tidak terjadi.

Gambar 4.7 menunjukkan hasil pengambilan data untuk kombinasi ukuran paket dan *packet injection rate* dengan *flow control* Ack / Nack *flow control*

Sedangkan pada *flow control* Ack, saat ukuran paket 8 – 10 flit terjadi *throughput* maksimum, namun saat ukuran paket diperbesar tidak terjadi kondisi saturasi melainkan *throughput* menurun. Hal tersebut disebabkan untuk jaringan dengan *flow control* Ack, flit dapat terus dikirimkan selama menerima Ack. Akan tetapi saat terjadi flit drop dan dikirimkan Nack harus dilakukan proses pengiriman ulang. Karena menggunakan *buffer* dengan ukuran 8 flit kemungkinan untuk terjaedinya flit drop dan proses pengiriman ulang semakin besar sehingga menyebabkan menurunnya *throughput* sesaat. Kemudian apabila ukuran paket kembali diperbesar, maka throughput akan cenderung mengalami kenaikan dan mengalami saturasi karena penggunaan buffer yang hanya berukuran 8 flit. Sehingga terjadinya flit drop dan congestion tidak merubah jumlah flit yang dapat diterima.



Gambar 4.7 Hasil Skenario 2 dengan *Flow Control* Ack



Gambar 4.8 Hasil Skenario 2 dengan *Flow Control* DyML

Gambar 4.8 menunjukkan hasil pengambilan data untuk kombinasi ukuran paket dan *packet injection rate* dengan *flow control* Dynamic Multi Level (DyML)

Pada *flow control* DyML dan Ack, saat ukuran paket 8 – 10 flit terjadi *throughput* maksimum, namun saat ukuran paket diperbesar tidak terjadi kondisi saturasi melainkan *throughput* menurun. Dimana proses pengiriman flit bergantung pada kondisi signalling 'STALL' dan 'GO' dan perubahan kondisi signalling bergantung pada *Buffer Fill Level* dan *Buffer Fluidity Level*. Sehingga dengan ukuran *buffer* yang hanya 8 flit, kemungkinan signalling berada pada kondisi 'STALL' akan semakin besar untuk terjadi yang menyebabkan *throughput* menurun. Kemudian apabila ukuran paket kembali diperbesar, maka throughput akan cenderung mengalami kenaikan hingga cenderung akan mengalami saturasi. Hal tersebut karena penggunaan buffer yang hanya berukuran 8 flit. Sehingga peningkatan jumlah flit pada jaringan dan terjadinya flit drop dan congestion tidak merubah jumlah flit yang dapat diterima.

Tabel 4.2 menunjukkan perbandingan kombinasi desain parameter ukuran paket dengan *packet injection rate* yang berbeda - beda untuk ketiga metode *flow control* yang digunakan dengan parameter yang diamati merupakan titik saturasi dan *throughput* maksimum.

Tabel 4.2 Perbandingan *Throughput* Maksimum dan Titik Saturasi pada Skenario 2

No	*Flow Control*	*Throughput* Maksimum	Titik Saturasi (Ukuran Paket)
1	Default	0.25742 flits / Cycle	8 – 10 flit
2	Stall / Go	Tidak Tersaturasi	Tidak Tersaturasi
3	Ack / Nack	Tidak Tersaturasi	Tidak Tersaturasi
4	Dynamic Multi Level	Tidak Tersaturasi	Tidak Tersaturasi

4.2 Analisis Pengaruh Parameter Desain dan *Flow Control* terhadap Performansi Jaringan

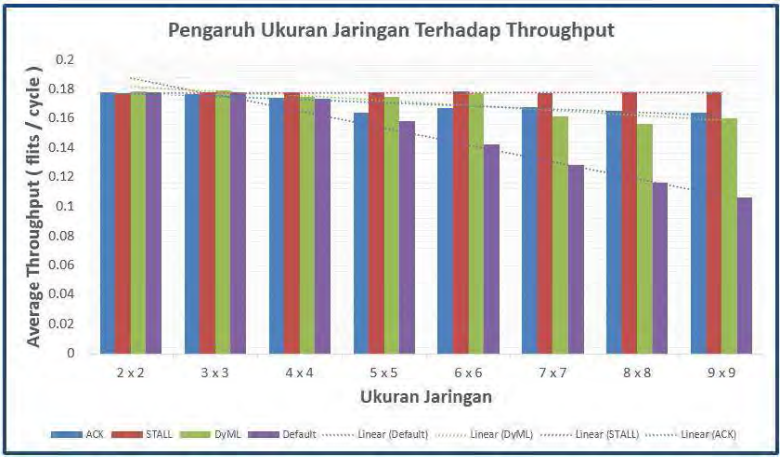
Pada *Network on Chip*, parameter desain seperti Ukuran Jaringan, Ukuran *Buffer*, Ukuran Paket, Algoritma routing, selection strategy, topologi, *flow control* dan *packet injection rate* dapat memberikan pengaruh pada parameter performansi seperti *throughput*, *delay*, reliability dan daya serta area dari jaringan yang digunakan. Dimana pengaruh dari parameter desain dapat meningkatkan atau menurunkan nilai dari parameter performansi tersebut.

Dengan menggunakan parameter desain yang telah ditentukan sebelumnya, maka dapat diperoleh hasil performansi berupa *throughput*, *delay*, daya yang digunakan, dan jumlah paket yang diterima pada jaringan dengan menggunakan masing – masing metode *flow control* Stall / Go, *flow control* Ack / Nack, dan *flow control* Dynamic Multi Level.*flow control* serta *flow control* default dari Noxim simulator tersebut.

4.2.1 Pengaruh Parameter Desain Terhadap *Throughput*

Parameter desain yang akan diuji pengaruhnya terhadap *throughput* adalah ukuran jaringan, *packet injection rate*, ukuran paket, ukuran *buffer*. Dengan parameter desain yang berubah – ubah nilainya, dilakukan perbandingan untuk metode *flow control* berbeda – beda yang digunakan.

Gambar 4.9 menunjukkan hasil pengaruh ukuran jaringan yang berubah – ubah terhadap *Throughput*



Gambar 4.9 Pengaruh Ukuran Jaringan Terhadap *Throughput*

Untuk jaringan dengan *flow control* default, semakin besar ukuran jaringan menyebabkan *throughput* menurun, perubahan *throughput* mulai menurun secara signifikan saat ukuran jaringan adalah 5 x 5. Hal tersebut dikarenakan *throughput* dihitung dalam satuan flits/cycle, dengan ukuran paket, ukuran *buffer* dan *packet injection rate* yang tetap apabila ukuran jaringan diperbesar maka waktu yang diperlukan oleh sebuah flit untuk mencapai tujuan akan bertambah sehingga menyebabkan *throughput* menurun. Selain itu metode pengiriman yang hanya mengizinkan mengirimkan 1 flit sebelum menerima ack menjadi salah satu faktor yang menurunkan *throughput* apabila ukuran jaringan diperbesar.

Jaringan yang menggunakan *flow control* Stall, dengan meningkatkan ukuran dari jaringan cenderung tidak mengalami perubahan yang signifikan, dimana rata – rata *throughput* yang dihasilkan sebesar 0.17803 flits/cycle. Pada teknik *flow control* stall, selama kanal signalling bertanda 'GO' maka flit dapat terus dikirimkan. Sehingga meskipun ukuran jaringan diperbesar tidak berpengaruh terhadap flit yang diteirma. Hal ini ditunjang dengan ukuran *buffer* yang cukup besar yaitu 8 flit dengan ukuran paket yang kecil yang hanya 2 – 4 flit dan pemlihan batas off sebesar 7 flit dan batas bawah 3 flit membuat kemungkinan kanal signalling bertanda 'STALL' semakin

kecil. Sehingga *throughput* tidak terpengaruh oleh meningkatnya ukuran jaringan.

Pada jaringan yang menerapkan *flow control* Ack, terjadi penurunan nilai *throughput* seiring dengan bertambahnya ukuran dari jaringan, dimana saat ukuran jaringan sebesar 5 x 5, terjadi penurunan *throughput* sebesar 0.0101 flits/cycle, kemudian saat ukuran jaringan semakin diperbesar *throughput* yang diberikan terjadi kenaikan dan penurunan dengan rata – rata *throughput* sebesar 0.165 flits / cycle. Hal tersebut dikarenakan untuk mengirimkan flit berikutnya, pengirim harus menerima sinyal Ack, selain itu apabila diterima sinyal Nack maka flit akan dikirimkan ulang. Sehingga dengan meningkatnya ukuran jaringan, dengan ukuran *buffer* dan jumlah yang dikirimkan tetap menyebabkan flit yang diterima per cycle akan berkurang.

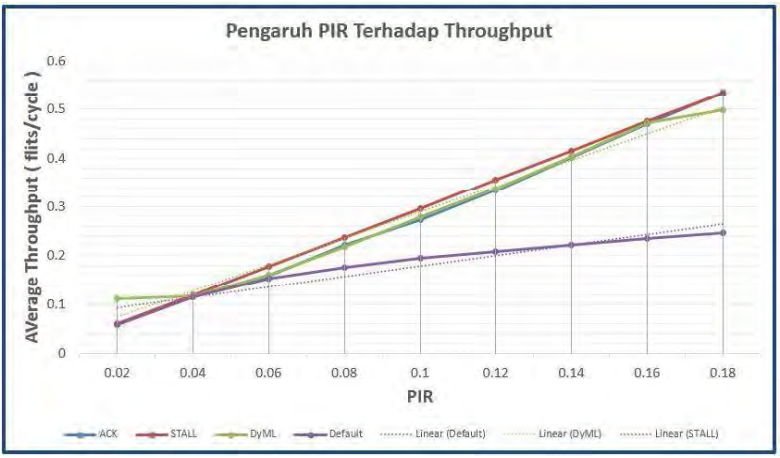
Sedangkan untuk jaringan dengan *flow control* Dynamic Multilevel, penurunan nilai *throughput* baru terjadi saat ukuran jaringan diperbesar hingga ke ukuran 7 x 7 dengan penurunan *throughput* sebesar 0.0158 flits/cycle. Hal tersebut dikarenakan kanal signalling akan bernilai 'STALL' sesuai dengan level dari *buffer* fill dan *buffer* fluidity, dimana apabila ukuran jaringan meningkat dan ukuran *buffer* serta jumlah flit yang dikirimkan bertambah, disaat kanal signalling sering terjadi 'STALL' maka pengirim akan lebih sering berhenti mengirimkan flit dimana berdampak pada menurunnya *throughput*.

Tabel 4.3 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran jaringan terhadap peningkatan *throughput* pada jaringan yang menggunakan *flow control* default

Tabel 4.3 Perbandingan Pengaruh Ukuran Jaringan Terhadap *Throughput*

Flow Control	*Throughput* (flits/cycle)		Peningkatan (%)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	0.147	0.106	-	-
Ack / Nack	0.169	0.163	14.96	53.77
Stall / Go	0.178	0.178	21.08	67.92
DyML	0.169	0.160	14.96	50.94

Sehingga dengan melakukan perbandingan peningkatan *throughput* rata – rata masing – masing *flow control* dan *Throughput* yang dicapai saat ukuran jaringan 9 x 9 dapat diperoleh *flow control*



Gambar 4.10 Pengaruh *Packet injection rate* Terhadap *Throughput*

yang memberikan peningkatan terbaik adalah *flow control* Stall / Go dengan peningkatan *throughput* rata – rata sebesar 21.08 % dan peningkatan througput maksimum sebesar 67.92% jika dibandingkan dengan jaringan yang menggunakan *flow control* default.

Gambar 4.10 adalah hasil pengaruh *packet injection rate* yang berubah – ubah terhadap *Throughput*

Bahwa *throughput* berbanding lurus dengan *packet injection rate* dimana semakin besar nilai dari *packet injection rate* maka *throughput* yang dihasilkan pun akan semakin meningkat. Untuk semua metode *flow control* yang digunakan pada jaringan sama – sama memberikan peningkatan pada *throughput* apabila *packet injection rate* ditingkatkan. Hal tersebut dikarenakan apabila *packet injection rate* diperbesar maka jumlah paket yang ada pada jaringan juga semakin banyak eshingga dapat meningkatkan throughput

Dengan ukuran *buffer* sebesar 8 flit, ukuran paket 2 – 4 flit, ukuran jaringan 5 x 5, *packet injection rate* akan berbanding lurus dengan *throughput*. *Flow control* stall / go memberikan peningkatan terbaik hal ini dikarenakan pada *flow control* stall / go flit akan terus dikirimkan selama kanal signalling 'GO'. Penggunaan batas off sebesar 7 flit pada bufer yang berukuran 8 flit dimana ukuran paket hanya 2 – 4 flit juga menjadi salah satu faktor yang menyebabkan probabilitas kanal

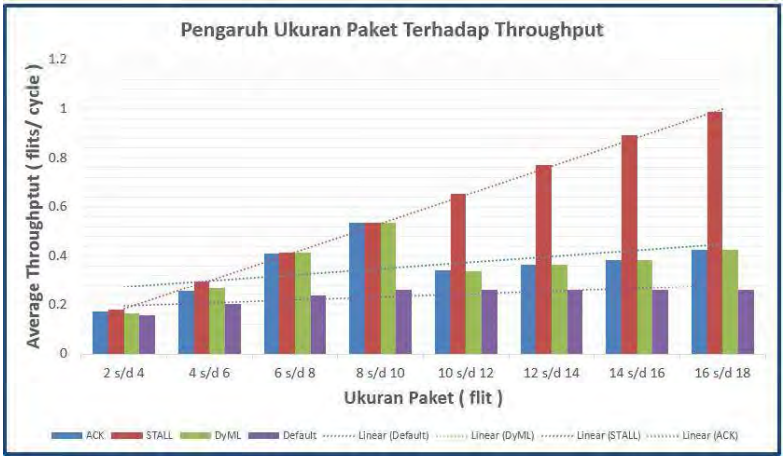
signalling 'STALL' menjadi cukup kecil. Sehingga flit dapat terus dikirimkan dan diterima oleh penerima.

Tabel 4.4 Perbandingan Pengaruh *Packet injection rate* Terhadap *Throughput*

Flow Control	*Throughput* (flits/cycle)		Peningkatan (%)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	0.179	0.246	-	-
Ack / Nack	0.285	0.534	59.21	117
Stall / Go	0.296	0.533	65.33	116
DyML	0.287	0.498	60.33	102

Sehingga dengan melakukan perbandingan peningkatan *throughput* rata **–** rata masing **–** masing *flow control* dan *Throughput* yang dicapai saat *packet injection rate* 0.18 dapat diperoleh *flow control* yang memberikan peningkatan terbaik adalah *flow control* Stall / Go dengan peningkatan *throughput* rata – rata sebesar 65.33 % dan *flow control* Ack/ Nack dengan peningkatan througput maksimum sebesar 117% jika dibandingkan dengan jaringan yang menggunakan *flow control* default

Gambar 4.11 menunjukkan hasil pengaruh ukuran paket yang berubah **–** ubah terhadap *Throughput*



Gambar 4.11 Pengaruh Ukuran Paket Terhadap *Throughput*

Bahwa *throughput* berbanding lurus dengan ukuran paket dimana semakin besar nilai dari ukuran paket yang digunakan maka *throughput* yang dihasilkan pun akan semakin meningkat. Tetapi tidak semua metode *flow control* yang digunakan mengalami kenaikan secara terus menerus.

Untuk jaringan yang menggunakan *flow control* default, saat ukuran paket yang digunakan adalah 10 - 12 flit, terjadi saturasi pada *throughput* yang dihasilkan dimana *throughput* maksimum nya bernilai 0.25746 flits/cycle. Hal ini dikarenakan penggunaan sliding window 1 untuk mengirimkan flit dan ukuran dari buffer yang tetap nilainya. Meskipun ukuran paket diperbesar, jumlah flit yang dapat diterima tidak akan bertambah banyak dan penambahan flit pada jaringan tidak akan mempengaruhi flit drop maupun Congestion lagi sehingga terjadi saturasi.

Sedangkan untuk jaringan yang menggunakan *flow control* Stall memberikan peningkatan *throughput* yang berbanding lurus dengan peningkatan ukuran paket, dimana kenaikan rata – rata *throughput*nya sebesar 0.1156 flits/cycle dengan laju kenaikan sebesar 0.0578 /flit. Adapun *throughput* maksimum yang dicapai saat ukuran paket 16 – 18 flit adalah sebesar 0.98744 flits/cycle dengan rata – rata *throughput* sebesar 0.5912 flits/cycle.

Jaringan dengan *flow control* Ack dan Dynamic Multilevel mememilki karakteristik yang hampir sama yaitu peningkatan ukuran paket hingga ukuran 8 – 10 flit menyebabkan peningkatan *throughput*, tetapi saat ukuran paket yang digunakan 10 – 12 flit terjadi penurunan *throughput* sesaat, kemudian terjadi peningkatan *throughput* kembali dengan laju kenaikan yang lebih kecil. Saat ukuran paket lebih kecil dari 10 flit, untuk jaringan dengan *flow control* Ack memiliki kenaikan rata – rata *throughput* sebesar 0.121524 flits/cycle dengan laju kenaikan sebesar 0.0607 / flit. Sedangkan dengan *flow control* DyML memiliki kenaikan rata – rata *throughput* sebesar 0.12377 flits/cycle dengan laju kenaikan sebesar 0.0618 / flit.

Penurunan *throughput* yang terjadi saat ukuran paket yang digunakan 10 – 12 flit untuk jaringan yang menggunakan *flow control* Ack adalah sebesar 0.19458 flits/cycle dan 0.19445 flits/cycle untuk *flow control* DyML.

Saat ukuran paket lebih besar dari 12 flit, untuk jaringan dengan *flow control* Ack memiliki kenaikan rata – rata *throughput* sebesar 0.029 flits/cycle dengan laju kenaikan sebesar 0.01456 / flit.

Sedangkan jaringan dengan *flow control* DyML, kenaikan rata – rata *throughput* sebesar 0.0285 flits/cycle dengan laju kenaikan sebesar 0.0142 / flit.

Terjadinya penurunan saat ukuran paket lebih besar dari 8 flit disebabkan karena ukuran *buffer* yang digunakan adalah sebesar 8 flit. Sehingga untuk jaringan dengan *flow control* Ack, saat ukuran flit yang diterima ukurannya besar dan *buffer* dalam kondisi full, maka akan terjadi flit drop, dan akan dilakukan transmisi ulang, akan tetapi apabila *buffer* tetap full maka flit akan di drop secara terus menerus sebelum *buffer* dapat menampung flit yang diterima. Sehingga hal tersebut akan mempengaruhi jumlah flit yang diterima sehingga *throughput* menurun.

Sedangkan pada *flow control* DyML, apabila dalam selang waktu tertentu router tidak mampu meneruskan flit atau tidak ada pergerakan pada *buffer* nya, maka *buffer* fluidity level akan semakin naik dan apabila *buffer* fluidity level mencapai level 5, kanal signalling akan bernilai 'STALL' terus sebelum *buffer* fluidity turun. Dimana penurunan *buffer* fluidity level dipengaruhi oleh flit yang diteruskan oleh *buffer*. Hal tersebut membuat jumlah flit yang diterima akan berkurang.

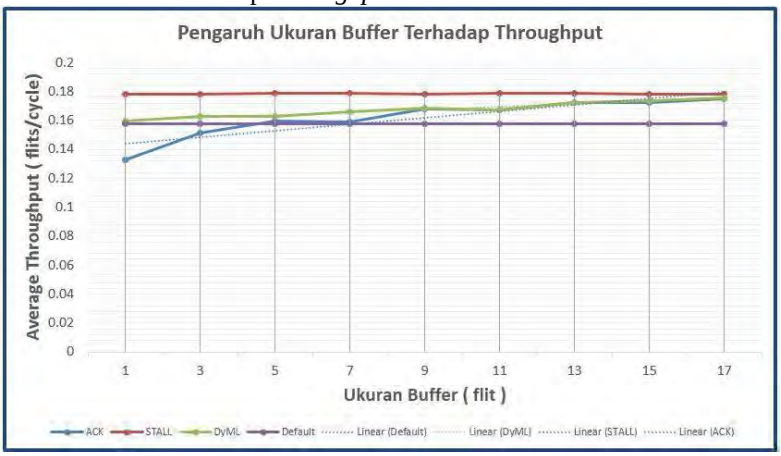
Tabel 4.5 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran paket terhadap peningkatan *throughput* pada jaringan yang menggunakan *flow control* default.

Tabel 4.5 Perbandingan Pengaruh Ukuran Paket Terhadap *Throughput*

Flow Control	*Throughput* (flits/cycle)		Peningkatan (%)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	0.235	0.257	-	-
Ack / Nack	0.360	0.427	53.19	66.14
Stall / Go	0.591	0.987	151	284
DyML	0.361	0.425	53.61	65.36

Sehingga dengan melakukan perbandingan peningkatan *throughput* rata – rata masing – masing *flow control* dan *Throughput* yang dicapai saat ukuran paket 16 – 18 flit dapat diperoleh *flow control* yang memberikan peningkatan terbaik adalah *flow control* Stall / Go dengan peningkatan *throughput* rata – rata sebesar 151 % dan dengan peningkatan througput maksimum sebesar 284% jika dibandingkan dengan jaringan yang menggunakan *flow control* default

Gambar 4.12 menunjukkanhasil pengaruh ukuran *buffer* yang berubah – ubah terhadap *Throughput*



Gambar 4.12 Pengaruh Ukuran *Buffer* Terhadap *Throughput*

Bahwa pengaruh dari perubahan ukuran *buffer* terhadap *throughput* untuk jaringan yang menggunakan *flow control* Stall dan default peningkatan dari ukuran *buffer* tidak berdampak yang cukup signifikan terhadap *throughput*, sedangkan untuk *flow control* Ack dan DyML peningkatan dari ukuran *buffer* memberikan peningkatan *throughput* yang berbanding lurus.

Jaringan yang menggunakan *flow control* default, peningkatan ukuran *buffer* tidak memberikan perubahan terhadap *throughput*. Hal ini dikarenakan pengiriman flit yang hanya berukuran 2 – 4 flit pada jaringan default hanya 1 flit saja, dengan ukuran buffer yang dirubah, jumlah flit yang diterima akan tetap karena pengiriman flit tidak bergantung pada ukuran buffer.

Begitu pula untuk jaringan yang menggunakan *flow control* Stall, peningkatan ukuran *buffer* tidak memberikan perubahan terhadap *throughput* yang cukup besar meskipun perubahan signalling GO dan STALL dipengaruhi oleh batas off dan batas on pada buffer tetapi dengan ukuran paket yang hanya 2 – 4 flit, tidak memberikan pengaruh yang besar serta kemungkinan jaringan berada pada kondisi STALL semakin kecil sehingga jumlah flit yang diterima tidak terlalu berubah secara signifikan.

Tetapi untuk jaringan yang menggunakan *flow control* Ack dan DyML, peningkatan ukuran *buffer* memberikan perubahan terhadap nilai dari *throughput*. Hal ini dikarenakan pada jaringan dengan Ack flit dikirimkan secara terus menerus selama menerima Ack dan meskipun menerima Nack maka flit akan dikirimkan meksipun merupakan transmisi ulang. Buffer berpengaruh terhadap flit drop akibat buffer full apabila terjadi congestion dan menyebabkan pengiriman Nack dan transmisi ulang. Untuk jaringan dengan DyML , buffer berpengaruh dikarenakan perubahan signalling bergantung kepada *buffer fluidity level* dan *buffer fill level*.

Tabel 4.6 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran *buffer* terhadap peningkatan *throughput* pada jaringan yang menggunakan *flow control* default.

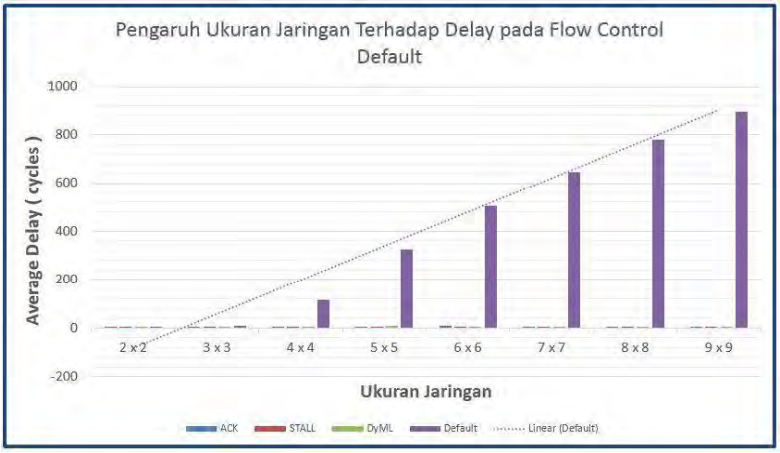
Tabel 4.6 Perbandingan Pengaruh Ukuran *Buffer* Terhadap *Throughput*

Flow Control	*Throughput* (flits/cycle)		Peningkatan (%)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	0.157	0.157	-	-
Ack / Nack	0.161	0.175	2.54	11.46
Stall / Go	0.178	0.178	13.37	13.37
DyML	0.167	0.175	6.36	11.46

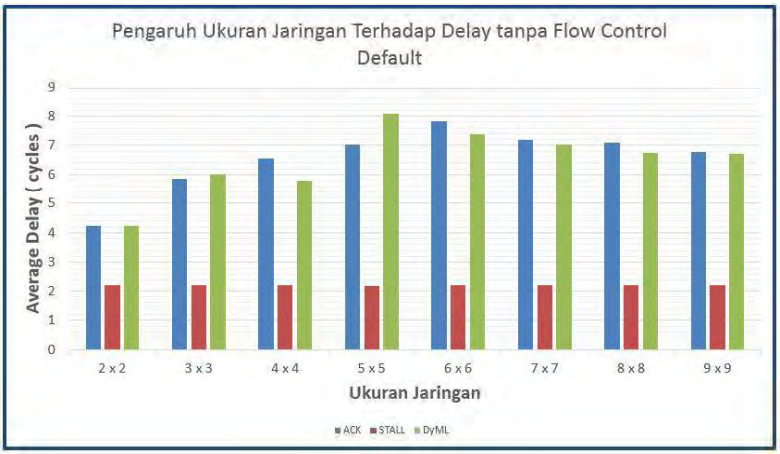
Sehingga dengan melakukan perbandingan peningkatan *throughput* rata – rata masing – masing *flow control* dan *Throughput* yang dicapai saat ukuran *buffer* 17 flit dapat diperoleh *flow control* yang memberikan peningkatan terbaik adalah *flow control* Stall / Go dengan peningkatan *throughput* rata – rata sebesar 13.37% dan dengan peningkatan througput maksimum sebesar 13.37% jika dibandingkan dengan jaringan yang menggunakan *flow control* default

4.2.2 Pengaruh Parameter Desain Terhadap *Delay*

Salah satu parameter performansi jaringan yang diamati adalah *delay* dimana dihitung dalam satuan cyclesdimana dilakukan pengujian dengan menggunakan 3 metode *flow control* untuk didapat pengaruhnya terhadap perubahan *delay*. Gambar 4.13 dan Gambar 4.14 menunjukkan hasil pengaruh ukuran jaringan yang berubah – ubah terhadap *delay* yang terjadi pada jaringan.



Gambar 4.13 Ukuran jaringan terhadap Delay dengan *Flow Control Default*



Gambar 4.14 Ukuran jaringan terhadap Delay tanpa *Flow Control Default*

Perbedaan *delay* yang sangat signifikan antara jaringan yang menggunakan *flow control* default dengan *flow control* lainnya dikarenakan untuk jaringan yang menggunakan *flow control* default hanya mengirimkan flit dengan sliding window 1, sedangkan *flow control* lainnya dapat mengirimkan flit secara terus menerus selama persyaratan yang diperlukan tercapai.

Untuk *flow control* Ack dan DyML seiring dengan bertambahnya jaringan nilai *delay* akan bertambah akan tetapi saat jaringan yang berukuran lebih besar dari 6 x 6 nilai *delay* cenderung tidak berubah. Hal terebut disebabkan karena dengan ukuran paket 2 – 4 flit dan ukuran *buffer* 8 flit serta *packet injection rate* 0.06, saat ukuran jaringan lebih besar dari 6 x 6, waktu yang dperlukan untuk mencapai tujuan tidak terlalu dipengaruhi oleh proses *buffer*ing dan routing pada router dikarenakan flit sudah tidak terlalu menumpuk pada beberapa router saja.

Sedangkan pada *flow control* Stall, perubahan ukuran jaringan cenderung tidak memberikan pengaruh yang signifikan terhadap perubahan *delay*. Hal ini dikarenakan karakteristik nya yang dapat mengirimkan flit terus – menerus selama signalling 'GO'. Meningkatnya ukuran jaringan justru akan memperkecil kemungkinan perubahan signalling menjadi 'STALL' semakin kecil.

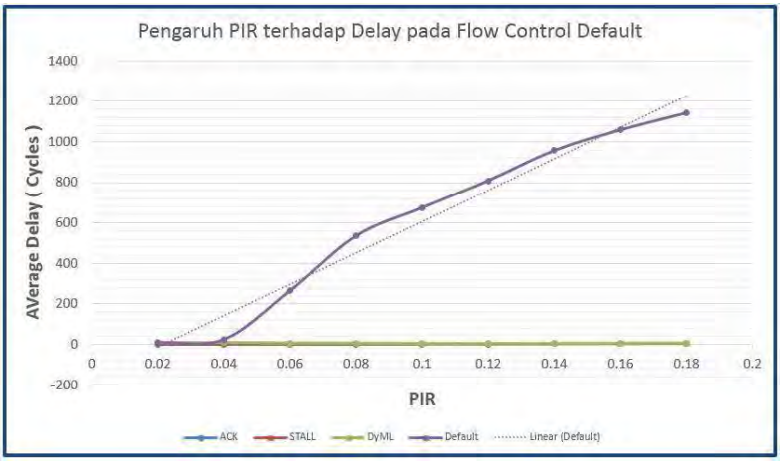
Tabel 4.7 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran jaringan terhadap perbaikan *delay* yang terjadi pada jaringan yang menggunakan *flow control* default.

Tabel 4.7 Perbandingan Pengaruh Ukuran Jaringan Terhadap *Delay*

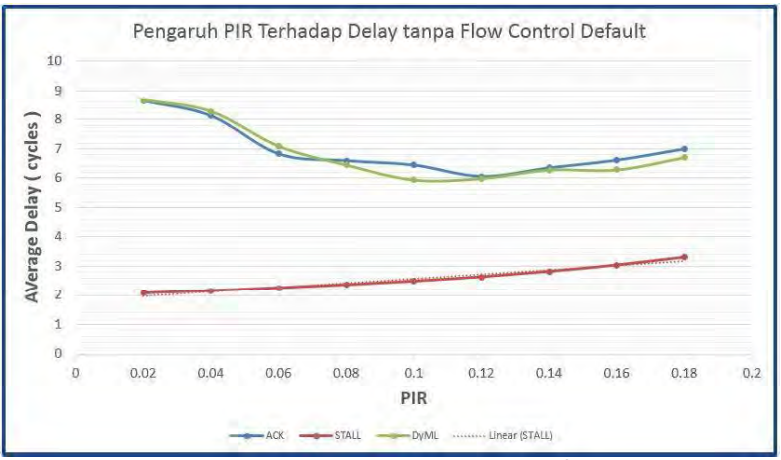
Flow Control	*Delay* (cycle)		Perbaikan (cycles)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	410.08	893.02	-	-
Ack / Nack	6.56	6.77	403.52	886.25
Stall / Go	2.23	2.24	407.85	890.78
DyML	6.49	6.70	403.59	886.32

Sehingga dengan melakukan perbandingan perbaikan *delay* rata – rata masing – masing *flow control* dan *delay* yang dicapai saat ukuran jaringan 9 x 9 dapat diperoleh *flow control* yang memberikan perbaikan terbaik adalah *flow control* Stall / Go dengan perbaikan *delay* rata – rata sebesar 407.85 cycles dan dengan perbaikan *delay* maksimum sebesar 890.78 cycles jika dibandingkan dengan jaringan yang menggunakan *flow control* default

Gambar 4.15 dan Gambar 4.16 menunjukkan hasil pengaruh *packet injection rate* yang berubah – ubah terhadap *delay* yang terjadi pada jaringan



Gambar 4.15 PIR terhadap Delay dengan *Flow Control Default*



Gambar 4.16 PIR terhadap Delay tanpa *Flow Control Default*

Untuk jaringan dengan *flow control* default, peningkatan dari *packet injection rate* memberikan kenaikan dari *delay*. Hal ini dikarenakan untuk jaringan yang menggunakan *flow control* default hanya mengirimkan flit dengan sliding window 1, sehingga saat jumlah flit yang dikirimkan ke jaringan meningkat, dengan ukuran buffer yang

tetap dapat menyebabkan congestion dan berakibat pada meningkatnya delay.

Jaringan yang mengugnakan *flow control* Stall kenaikan nilai *delay* yang terjadi seiring dengan *packet injection rate* yang ditingkatkan hal ini dikarenakan dengan ukuran *buffer* yang tetap sebsear 8 flit saat intensitas flit yang dikirimkan meningkat akan terjadi kecenderungan signallling berubah menjadi 'STALL' yang menyebabkan pengiriman flit pada semua router berhenti. Hal tersebut menyebabkan meningkatnya *delay*.

Untuk jaringan yang menggunakan *flow control* Ack mengalami penurunan *delay* apabila *packet injection rate* ditingkatkan hingga 0.12 dengan *delay* terendah sebesar 6.056, kemudian apabila *packet injection rate* ditingkatkan, *delay* pada jaringan akan meningkat. Saat *packet injection rate* bernilai 0.18 maka *delay* yang dicapai pada jaringan adalah sebsar 6.998 cycles. Penurunan *delay* saat *packet injection rate* bernilai lebih kecil dari 0.12 dikarenakan saat itu proses pengiriman flit sedang optimal sehingga Congestion dan antrian tidak terlalu mempengaruhi proses pengiriman flit. Akan tetapi hal yang sebaliknya terjadi saat packet injectionr ate dipebesar hingga lebih dari 0.12.

Sedangkan untuk jaringan yang menggunakan *flow control* DyML mengalami penurunan *delay* apabila *packet injection rate* ditingkatkan hingga 0.12 dengan *delay* terendah sebesar 5.98757, kemudian apabila *packet injection rate* ditingkatkan, *delay* pada jaringan akan meningkat. Saat *packet injection rate* benrilai 0.18 maka *delay* yang dicapai pada jaringan adalah sebesar 6.282 cycles. Penurunan *delay* hingga *packet injection rate* 0.12 dikarenakan dengan *packet injection rate* lebih kecil dari 0.12, proses pengiriman flit dengan ukuran *buffer* sebesar 8 flit justru maksimal tanpa terjadi Congestion dan antrian yang berlebihan yang dapat menyebabkan signalling 'STALL' lebih lama dibanding 'GO'. Oleh karena itu *delay* justru menurun. Akan tetapi saat *packet injection rate* lebih besar dari 0.12, proses pengiriman flit sudah tidak optimal dan signalling 'STALL' cenderung lebih sering terjadi dibanding 'GO' sehingga *delay* akan meningkat.

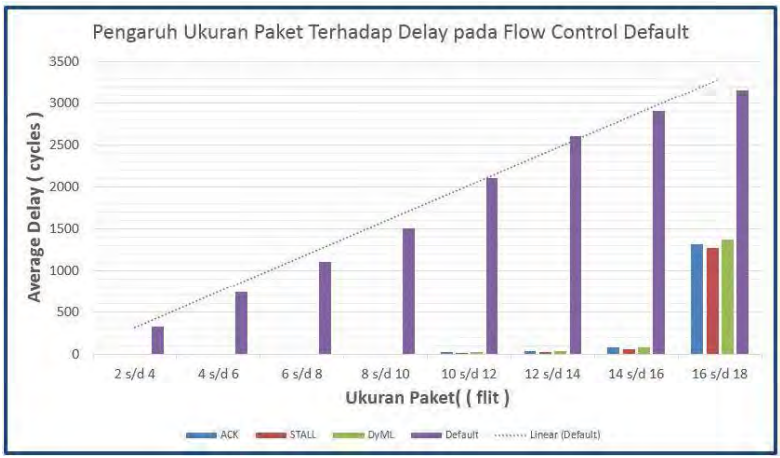
Tabel 4.8 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan *packet injection rate* terhadap perbaikan *delay* yang terjadi pada jaringan yang menggunakan *flow control* default.

Tabel 4.8 Perbandingan Pengaruh PIR Terhadap *Delay*

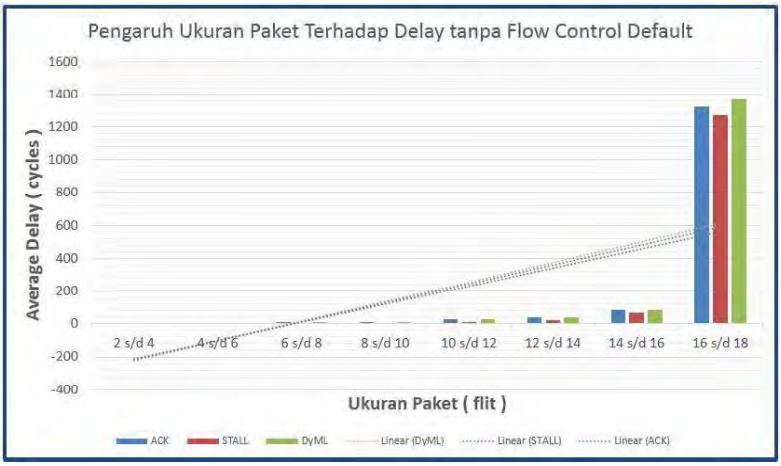
Flow Control	*Delay* (cycle)		Perbaikan (cycles)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	608.59	1142.2	-	-
Ack / Nack	6.96	6.99	601.63	1135.21
Stall / Go	2.56	3.3	606.03	1138.9
DyML	6.85	6.70	601.74	1135.5

Sehingga dengan melakukan perbandingan perbaikan *delay* rata – rata masing – masing *flow control* dan *delay* yang dicapai saat ukuran *packet injection rate* 0.18 dapat diperoleh *flow control* yang memberikan perbaikan terbaik adalah *flow control* Stall / Go dengan perbaikan *delay* rata – rata sebesar 606.03 cycles dan dengan perbaikan *delay* maksimum sebesar 1138.9 cycles jika dibandingkan dengan jaringan yang menggunakan *flow control* default

Gambar 4.17 dan Gambar 4.18 menunjukkan hasil pengaruh ukuran paket yang berubah – ubah terhadap *delay* yang terjadi pada jaringan



Gambar 4.17 Ukuran paket terhadap Delay dengan *Flow Control Default*



Gambar 4.18 Ukuran Paket terhadap Delay tanpa *Flow Control Default*

Bahwa semakin besar ukuran paket yang digunakan maka *delay* yang terjadi pada jaringan akan semakin meningkat. Dari ketiga *flow control* yang diterapkan, memberikan hasil dengan karakteristik yang sama.

Semakin meningkatnya ukuran paket akan berdampak pada meningkatnya jumlah flit yang dikirimkan. Apabila dengan ukuran *buffer* sebesar 8 flit dan *packet injection rate* 0.06, semakin besar ukuran paket akan menyebabkan semakin lama waktu yang diperlukan oleh flit untuk sampai ke tujuan apabila menggunakan *flow control* default yang hanya mengirimkan flit dengan sliding window 1.
Sedangkan pada jaringan dengan *flow control* Ack akan menyebabkan kemungkinan untuk terjadinya antrian yang berlebihan hingga flit drop yang menimbulkan pengiriman ulang semakin besar.

Pada jaringan dengan *flow control* Dynamic Multilevel dan Stall / Go, semakin besar ukuran paket akan membuat kemungkinan kondisi signalling 'STALL' semakin tinggi yang menyebabkan *delay* akan bertambah besar.

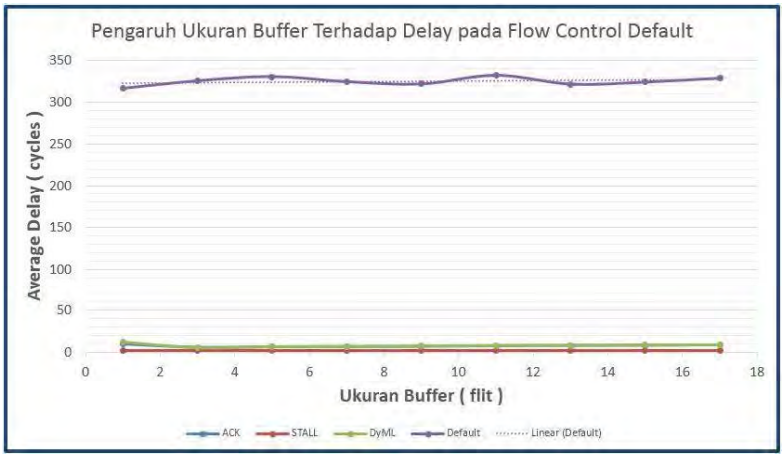
Tabel 4.9 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran paket terhadap perbaikan *delay* yang terjadi pada jaringan yang menggunakan *flow control* default.

Tabel 4.9 Perbandingan Pengaruh Ukuran Paket Terhadap *Delay*

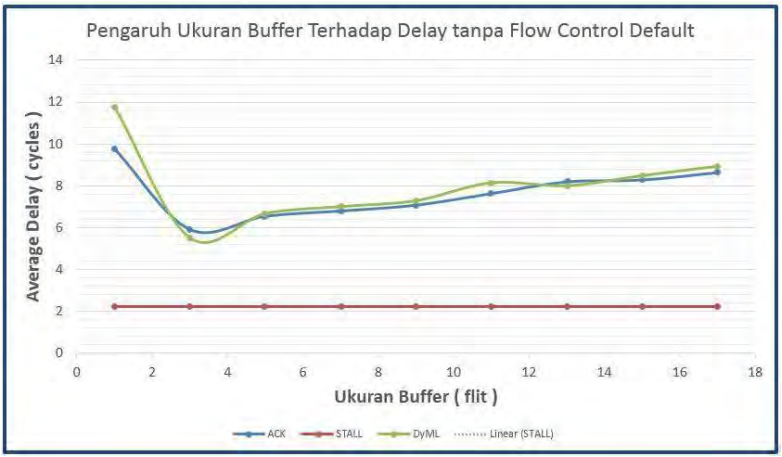
Flow Control	*Delay* (cycle)		Perbaikan (cycles)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	1805.56	3159.98	-	-
Ack / Nack	187.93	1320.22	1617.63	1839.76
Stall / Go	173.61	1272.16	1631.95	1887.82
DyML	194.21	1369.50	1611.35	1790.48

Sehingga dengan melakukan perbandingan perbaikan *delay* rata – rata masing – masing *flow control* dan *delay* yang dicapai saat ukuran paket 16 – 18 flit dapat diperoleh *flow control* yang memberikan perbaikan terbaik adalah *flow control* Stall / Go dengan perbaikan *delay* rata – rata sebesar 1631.95 cycles dan dengan perbaikan *delay* maksimum sebesar 1887.82 cycles jika dibandingkan dengan jaringan yang menggunakan *flow control* default

Gambar 4.19 dan Gambar 4.20 menunjukkan hasil pengaruh ukuran *buffer* yang berubah – ubah terhadap *delay* yang terjadi pada jaringan



Gambar 4.19 Ukuran Buffer terhadap Delay dengan *Flow Control Default*



Gambar 4.20 Ukuran Buffer terhadap Delay tanpa *Flow Control Default*

Perubahan dari ukuran *buffer* tidak memberikan pengaruh secara signifikan terhadap perubahan *delay* untuk jaringan yang menerapkan *flow control* default dan *flow control* Stall.

Hal ini dikarenakan untuk jaringan yang menggunakan *flow control* default, *buffer* tidak memberikan dampak yang signifikan dikarenakan proses pengiriman flit yang hanya dengan sliding window 1. Sedangkan pada *flow control* Stall, proses perubahan signalling 'STALL' dan 'GO' dipengaruhi oleh batas off dan batas on. Dimana batas off dan batas on berubah – ubah mengikuti ukuran *buffer*.

Sedangkan untuk jaringan yang menerapkan *flow control* Ack dan DyML memiliki karakteristik yang sama yaitu untuk ukuran *buffer* bernilai 1 *delay* yang terjadi cukup besar, kemudian untuk *buffer* bernilai 3 *delay* yang terjadi pada jaringan mengalami penurunan. Selanjutnya apabila *buffer* diperbesar lebih dari 3 flit, maka *delay* yang terjadi pada jaringan berbanding lurus.

Untuk *flow control* DyML, pengiriman flit bergantung pada 'STALL' dan 'GO' signalling yang bergantung pada *buffer* fluidity level dan *buffer* fill level. Sehingga saat ukuran *buffer* hanya lah 1, maka kemungkinan untuk jaringan berada pada kondisi 'GO' sangat kecil sehingga *delay* yang diberikan besar. Sedangkan untuk jaringan dengan *flow control* Ack, ukuran *buffer* yang hanya sebesar 1 flit menyebabkan kemungkinan terjadi flit drop semakin besar yang mengakibatkan

kemungkinan terjadinya pengiriman ulang semakin besar. Apabila terjadi pengiriman ulang dengan ukuran *buffer* yang tetap hanya 1 flit tidak menutup kemungkinan untuk terjadi flit drop kembali yang akan berujung pada meningkatnya nilai *delay* pada jaringan.

Tabel 4.10 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran *buffer* terhadap perbaikan *delay* yang terjadi pada jaringan yang menggunakan *flow control* default.

Tabel 4.10 Perbandingan Pengaruh Ukuran *Buffer* Terhadap *Delay*

Flow Control	*Delay* (cycle)		Perbaikan (cycles)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	324.83	328.52	-	-
Ack / Nack	7.64	8.64	317.19	319.88
Stall / Go	2.24	2.24	322.59	326.28
DyML	7.97	8.92	316.86	319.6

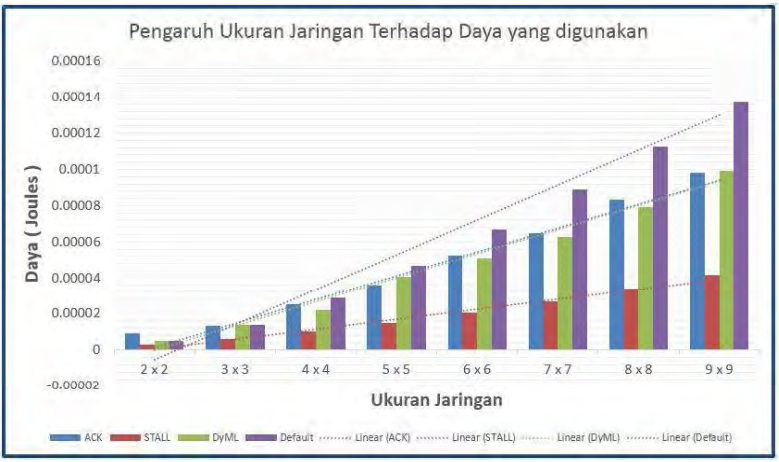
Sehingga dengan melakukan perbandingan perbaikan *delay* rata **–** rata masing **–** masing *flow control* dan *delay* yang dicapai saat ukuran *buffer* 17 flit dapat diperoleh *flow control* yang memberikan perbaikan terbaik adalah *flow control* Stall / Go dengan perbaikan *delay* rata **–** rata sebesar 322.59 cycles dan dengan perbaikan *delay* maksimum sebesar 326.28 cycles jika dibandingkan dengan jaringan yang menggunakan *flow control* default

Perbedaan *delay* yang sangat signifikan antara jaringan yang menggunakan *flow control* default dengan *flow control* lainnya dikarenakan untuk jaringan yang menggunakan *flow control* default hanya mengirimkan flit dengan sliding window 1, sedangkan *flow control* lainnya dapat mengirimkan flit secara terus menerus selama persyaratan yang diperlukan tercapai.

4.2.3 Pengaruh Parameter Desain Terhadap Daya yang Digunakan

Dalam suatu jaringan terdapat banyak komponen dan faktor yang dapat mempengaruhi penggunaan daya seperti algoritma routing, fungsi selection yang digunakan serta kondisi router.

Gambar 4.21 menunjukkan hasil pengaruh ukuran jaringan yang berubah **–** ubah terhadap penggunaan daya pada jaringan



Gambar 4.21 Grafik Pengaruh Ukuran Jaringan Terhadap Daya

Sehingga dengan meningkatkan ukuran jaringan akan berdampak pada meingkatnya pula penggunaan daya pada jaringan baik yang menerapkan *flow control* default maupun yang menerapkan metode *flow control* lainnya

Penggunaan daya pada jaringan dipengaruhi oleh durasi pengiriman flit dari sumber ke tujuan, jumlah router yang dilalui, dan juga proses routing. Sehingga saat menggunakan *flow control* stall / go dengan ukuran jaringan yang ditingkatkan, flit lebih cepat dikirimkan ke node tujuan saat signaling 'GO'. Flit juga lebih sedikit yang berada di *buffer* karena adanya batas off sehingga daya yang digunakan untuk proses pada router lebih sedikit apabila dibandingkan dengan jaringan yang menerapkan flow conrol DyML. Dimana pada DyML, harus dilakukan pengecekan secara berkala mengenai jumlah pop yang terjadi pada *buffer* dan tingkat kemungkinan terjadinya 'STALL' pada DyML cukup tinggi. Sedangkan pada Ack dan Default flit yang dapat dikirimkan hanya 1 sebelum menerima Ack balikan sehingga saat ukuran diperbesar, saat terjadi flit drop maka penggunaan daya meningkat.

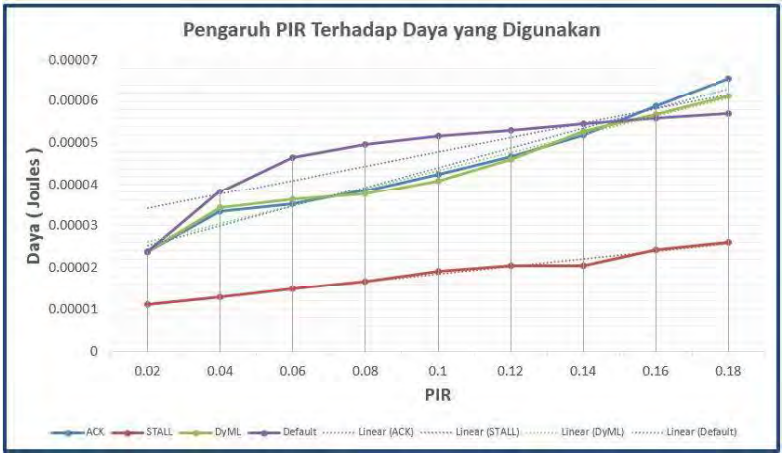
Tabel 4.11 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran jaringan terhadap perbaikan penggunaan daya pada jaringan yang menggunakan *flow control* defaul

Tabel 4.11 Perbandingan Pengaruh Ukuran Jaringan Terhadap Daya

Flow Control	Penggunaan Daya (nJoules)		Perbaikan (%)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	62.632	137.509	-	-
Ack / Nack	47.884	98.358	23.54	28.47
Stall / Go	19.618	41.577	68.67	69.76
DyML	46.716	99.164	25.41	27.88

Sehingga dengan melakukan perbandingan penggunaan daya maksimum rata – rata masing – masing *flow control* dan daya yang dicapai saat ukuran jaringan 9 x 9 dapat diperoleh *flow control* yang memberikan perbaikan terbaik adalah *flow control* Stall / Go dengan perbaikan penggunaan daya rata – rata sebesar 68.67% dan perbaikan penggunaan daya maksimum sebesar 69.76% jika dibandingkan dengan jaringan yang menggunakan *flow control* default.

Gambar 4.21 menunjukkan hasil pengaruh *packet injection rate* yang berubah – ubah terhadap penggunaan daya pada jaringan



Gambar 4.22 Grafik Pengaruh PIR Terhadap Daya

Bahwa *packet injection rate* memberikan pengaruh yang berbanding lurus dengan penggunan daya. Dimana dengan meningkatkan *packet injection rate* maka daya yang digunakan juga

semakin meningkat. Untuk ketiga metode *flow control* yang dibandingkan dengan metode *flow control* default, secara keseluruhan menunjukkan karakteristik yang sama.

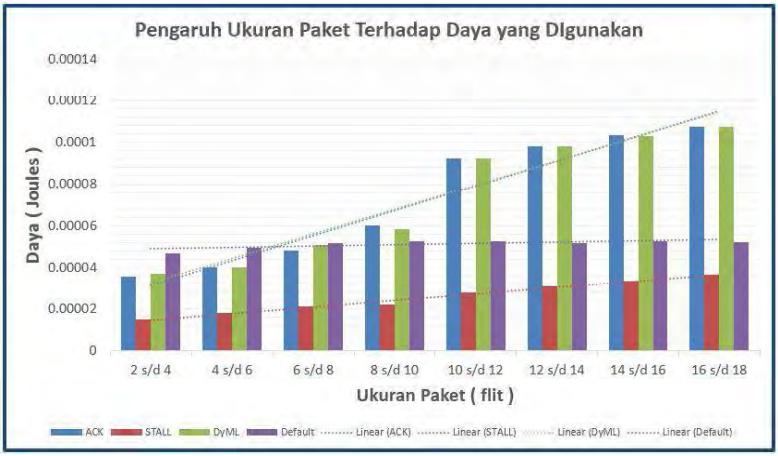
Packet injection rate berpengaruh pada jumlah flit yang dikirimkan dimana jumlah flit berbanding lurus dengan penggunaan daya. Jumlah flit pada jaringan juga memberikan dampak pada kelancaran proses routing dan kecepatan proses routing. Pada jaringan dengan *flow control* default, dimana proses penerimaan dan pengiriman flit hanya dapat 1 flit saja denga meningkatnya *packet injection rate* dengan ukuran *buffer* 8 flit dan ukuran paket 2 – 4 flit dapat menyebabkan antrian pada *buffer* dan meningkatnya waktu pengiriman flit ke tujuan. Begitu pula dengan *flow control* Ack, dengan meningkatnya jumlah flit yang dikirimkan meningkatkan kemungkinan terjadinya flit drop yang menyebabkan proses transmisi ulang sehingga menambah penggunaan daya. Berbeda dengan *flow control* Stall dimana selama kanal signalling 'GO' flit akan terus dikirimkan. Sehingga meningkatnya *packet injection rate* tidak memiliki pengaruh yang terlalu besar pada proses routing, *buffer*ing dan proses pengiriman ke node tujuan sehingga penggunaan daya dengan menggunakan *flow control* Stall lebih sedikit.

Tabel 4.12 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan *packet injection rate* terhadap perbaikan penggunaan daya pada jaringan yang menggunakan *flow control* default

Tabel 4.12 Perbandingan Pengaruh PIR Terhadap Daya

Flow Control	Penggunaan Daya (nJoules)		Perbaikan (%)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	47.846	57.093	-	-
Ack / Nack	44.118	65.395	7.79	-14.54
Stall / Go	18.501	26.103	61.33	54.27
DyML	43.383	61.344	9.32	- 7.44

Sehingga dengan melakukan perbandingan penggunaan daya maksimum rata – rata masing – masing *flow control* dan daya yang dicapai saat *packet injection rate* 0.18 dapat diperoleh *flow control* yang memberikan perbaikan terbaik adalah *flow control* Stall / Go dengan perbaikan penggunaan daya rata – rata sebesar 61.33% dan perbaikan



Gambar 4.23 Grafik Pengaruh Ukuran Paket Terhadap Daya

penggunaan daya maksimum sebesar 54.27% jika dibandingkan dengan jaringan yang menggunakan *flow control* default.

Gambar 4.23 menunjukkan hasil pengaruh ukuran paket yang berubah – ubah terhadap penggunaan daya pada jaringan

Untuk jaringan yang menggunakan *flow control* default, saat ukuran paket yang digunakan adalah 8 - 10 flit mulai terjadi saturasi pada daya yang digunakan. Hal tersebut berkaitan dengan jumlah flit yang dapat diterima dan waktu pemrosesan dari flit pada jaringan

Sedangkan untuk jaringan yang menggunakan *flow control* Stall memberikan peningkatan daya yang digunakan yang berbanding lurus dengan peningkatan ukuran paket. Hal tersebut dikarenakan saat ukuran paket semakin besar maka kemungkinan terjadinya perubahan signalling menjadi STALL akibat buffer penerima penuh semakin besar. Apabila signalling STALL , maka waktu yang diperlukan untuk pemrosesan dan pengiriman flit semakin lama yang berakibat pada meningkatnya konsumsi daya.

Jaringan dengan *flow control* Ack dan Dynamic Multilevel mememilki karakteristik yang hampir sama yaitu peningkatan ukuran paket hingga ukuran 8 – 10 flit menyebabkan peningkatan penggunaan daya, tetapi saat ukuran paket yang digunakan 10 – 12 flit terjadi peningkatan penggunaan daya sesaat yang cukup signifikan, kemudian

terjadi peningkatan penggunaan daya kembali dengan laju kenaikan yang lebih kecil.

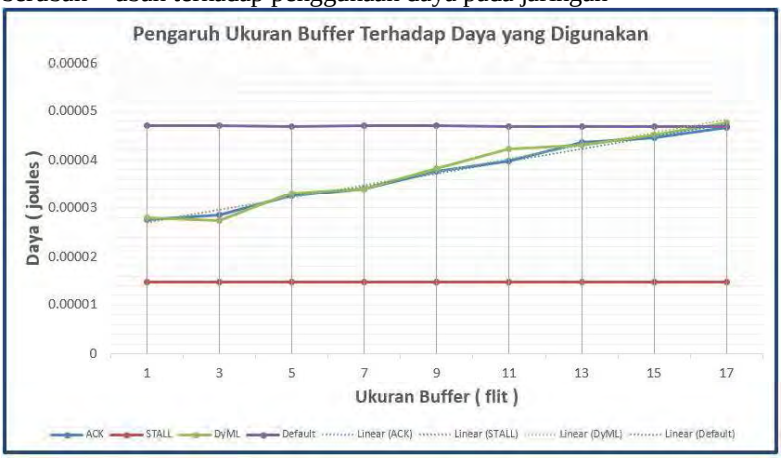
Tabel 4.13 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran paket terhadap penggunaan daya pada jaringan yang menggunakan *flow control* default

Tabel 4.13 Perbandingan Pengaruh Ukuran Paket Terhadap Daya

Flow Control	Penggunaan Daya (nJoules)		Perbaikan (%)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	51.259	52.302	-	-
Ack / Nack	73.156	107.546	- 42.71	-105
Stall / Go	25.664	36.840	49.93	29.5
DyML	73.440	107.532	43.27	- 105

Sehingga dengan melakukan perbandingan penggunaan daya maksimum rata – rata masing – masing *flow control* dapat diperoleh *flow control* yang memberikan perbaikan terbaik adalah *flow control* Stall / Go dengan perbaikan penggunaan daya rata – rata sebesar 49.93% dan penggunaan daya maksimum sebesar 29.5% jika dibandingkan dengan jaringan yang menggunakan *flow control* default.

Gambar 4.24 menunjukkan hasil pengaruh ukuran *buffer* yang berubah – ubah terhadap penggunaan daya pada jaringan



Gambar 4.24 Grafik Pengaruh Ukuran *Buffer* Terhadap Daya

Dimana untuk jaringan yang menggunakan *flow control* Stall dan default peningkatan dari ukuran *buffer* tidak berdampak yang cukup signifikan terhadap penggunaan daya, sedangkan untuk *flow control* Ack dan DyML peningkatan dari ukuran *buffer* memberikan peningkatan penggunaan daya yang berbanding lurus.

Ukuran *buffer* berkaitan dengan jumlah flit yang dapat antri pada router dan mempengaruhi besar kecilnya kemungkinan terjadinya contenction dan flit drop. Pada jaringan dengan *flow control* default, ukuran *buffer* tidak memberikan pengaruh yang cukup besar dikarenakan proses pengiriman flit yang hanya 1 flit saja membuat ukuran *buffer* tidak terlalu berpengaruh dengan *packet injection rate* yang 0.06 dan ukuran paket yang 2 – 4. Begitu pula dengan *flow control* Ack, akan tetapi untuk *flow control* ada proses retransmisi untuk proses pengiriman yang gagal. Ukuran *buffer* memberikan pengaruh yang berbanding lurus dengan penggunaan daya dikarenakan pada Ack flit dikirimkan secara terus menerus, sehingga ukuran *buffer* berdampak pada daya yang diperlukan untuk proses *buffer*ing dan flit yang standby di router.

Buffer memberikan pengaruh pada pegantian signalling 'GO' dan 'STALL' pada *flow control* DyML dan Stall dengan penentuan batas Off dan batas On tertentu. *flow control* Stall memberikan hasil penggunaan daya yang terendah dan cenderung tidak dipengaruhi oleh ukuran *buffer* dikarenakan ukuran paket yang digunakan hanya 2 – 4 flit sedangkan proses perubahan signalling antara 'STALL' dan 'GO' lebih cepat sehingga waktu yang diperlukan untuk berada di router dan proses *buffer*ing cenderung lebih cepat.

Tabel 4.14 menunjukkan hasil perbandingan pengaruh penggunaan *flow control* dan ukuran *buffer* terhadap perbaikan penggunaan daya pada jaringan yang menggunakan *flow control* default

Tabel 4.14 Perbandingan Pengaruh Ukuran *Buffer* Terhadap Daya

Flow Control	Penggunaan Daya (nJoules)		Perbaikan (%)	
	Rata–Rata	Maksimum	Rata – Rata	Maksimum
Default	46.947	46.870	-	-
Ack / Nack	37.230	46.740	20.69	0.27
Stall / Go	14.916	14.927	68.22	68.15
DyML	37.662	47.558	19.77	- 1.4

Sehingga dengan melakukan perbandingan penggunaan daya maksimum rata – rata masing – masing *flow control* dan daya yang dicapai saat ukuran *buffer* 17 flit dapat diperoleh *flow control* yang memberikan perbaikan terbaik adalah *flow control* Stall / Go dengan perbaikan penggunaan daya rata – rata sebesar 68.22% dan penggunaan daya maksimum sebesar 68.15% jika dibandingkan dengan jaringan yang menggunakan *flow control* default.

4.2.4 Pengaruh Parameter Desain Terhadap Perubahan Parameter Performansi Jaringan

Dengan merubah nilai dari parameter desain seperti *packet injection rate*, ukuran paket ,ukuran *buffer* dan ukuran jaringan dapat mempengaruhi parameter performansi jaringan seperti *throughput*, *delay* dan daya yang digunakan.

Dengan berdasarkan data hasil percobaan pada sub bab sebelumnya maka untuk dapat meningkatkan performansi jaringan dapat melakukan perubahan pada parameter desain seperti yang ditampilkan pada Tabel 4.15

Tabel 4.15 Pengaturan Parameter Desain terhadap Parameter Performansi Jaringan

Parameter Performansi Jaringan	Pengaturan Parameter Desain			
	Packet injection rate	Ukuran Paket	Ukuran *Buffer*	Ukuran Jaringan
Throughput	Diperbesar	Diperbesar	Diperbesar	Diperbesar
Delay	Diperkecil	Diperkecil	Diperbesar	Diperkecil
Daya	Diperkecil	Diperkecil	Diperkecil	Diperkecil

Untuk meningkatkan nilai dari *throughput packet injection rate* harus diperbesar dikarenakan semakin tinggi nilai dari *packet injection rate* maka akan semakin sering paket dikirimkan pada jaringan. Sehingga dengan ukuran *buffer* dan ukuran paket yang sesuai sehingga tidak terjadi Congestion pada jaringan, dengan memperbanyak paket yang dikirim akan meningkatkan throuhgput. Selain itu meskipun terjadi Congestion atau packet drop pada jaringan, dengan meningkatkan *packet injection rate* akan mempercepat proses pengiriman ulang paket. Selain itu dengan memperbesar ukuran paket akan mengakibatkan jumlah flit yang dikirimkan akan semakin banyak. *Throughput* dinyatakan dengan satuan flits/cycle. Sehinggag dengan semakin banyak

flit yang dikirimkan dan semakin banyak flit yang diterima akan meningkatkan *throughput*. Peningkatan ukuran dari *buffer* dapat meningkatkan *throughput* untuk jaringan dengan trafik yang padat sehingga flit yang diterima di router dapat disimpan pada *buffer* sebanyak – banyaknya dan mencegah untuk terjadinya flit drop. Sedangkan dengan memperbesar ukuran jaringan akan berakibat pada bertambahnya penyebaran dari flit dan mengurangi kemungkinan terjadinya Congestion akibat *buffer* full atau kanal yang full sehingga dapat meningkatkan jumlah flit yang diteirma dan meningkatkan *throughput*.

Untuk jaringan dengan ukuran *buffer* yang terbatas dan dengan ukuran jaringan yang besar, dengan memperkecil nilai dari *packet injection rate* yang menyebabkan paket yang dikirimkan per cycle nya akan berkurang dapat memperkecil nilai dari *delay* hal ini disebabkan dengan berkurangnya paket yang dikirimkan per cycle, akan mengurangi antrian pada router yang menyebabkan *delay* akan berkurang. Hal tersebut berkaitan dengan meningkatkan ukuran dari *buffer*, semakin besar ukuran dari *buffer* akan menyebabkan kemungkinan untuk terjadinya antrian pada *buffer* berkurang sehingga flit akan diterima dengan *delay* yang rendah. Selain itu dengan memperkecil ukuran dari paket menyebabkan flit yang dikirimkan akan berkurang sehingga kemungkinan terjadinya antrian berkurang sehingga *delay* pun berkurang. Selain itu untuk mengurangi *delay* dapat dilakukan dengan memperkecil ukuran dari jaringan. Hal ini dikarenakan jarak atau hop yang perlu dilewati oleh sebuah hop untuk mencapai tujuan akan semakin pendek.

Untuk mengurangi penggunaan daya pada jaringan dapat dilakukan dengan mengurangi ukuran dari *buffer* dan ukuran dari jaringan. Hal ini dikarenakan selama flit tersimpan di *buffer* dan waktu yang diperlukan flit untuk mencapai tujuan mempengaruhi penggunaan daya. Sehingga dengan memperkecil *buffer* menyebebkan proses *buffer*ing flit berkurang begitu pula dengan memperkecil ukuran jaringan sehingga waktu yang diperlukan untuk mencapai tujuan akan berkurang yang berakibat pada daya yang diperlukan akan berkurang. Selain itu dengan mengurangi ukuran paket dan *packet injection rate* berkaitan dengan jumlah flit yang dikirimkan pada jaringan. Proses penerimaan flit, penentuan port output, proses routing dan penerusan flit memerlukan daya, sehingga apabila flit yang dikirimkan berkurang akan menyebabkan daya yang digunakan akan berkurang juga.

Dari percobaan dengan menggunakan beberapa parameter desain yang dibuat tetap yaitu simulation time 100000 cycles, traffic transpose, routing xy, ukuran paket 2 – 4 flit, ukuran *buffer* 8 flit, *packet injection rate* 0.06 dan ukuran jaringan 5 x 5 mesh dapat diperoleh perubahan parameter desain yang memberikan pengaruh terbesar pada perubahan nilai dari parameter performansi jaringan *throughput*, *delay* dan daya yang digunakan seperti yang ditunjukkan pada Tabel 4.16, Tabel **4.17**, Tabel 4.18 dan Tabel 4.19.

Tabel 4.16 Pengaruh Perubahan Parameter Desain dengan Default *Flow Control*

Default *Flow Control*				
Parameter Performansi	Parameter Desain			
	Ukuran Jaringan	*Packet injection rate*	Ukuran Paket	Ukuran *Buffer*
Throughput	Ketiga	Pertama	Kedua	Keempat
Delay	Ketiga	Kedua	Pertama	Keempat
Daya	Pertama	Kedua	Ketiga	Keempat

Pada jaringan tanpa flow control, perubahan dari throughput sangat dipengaruhi oleh perubahan pada *packet injection rate* .Hal ini dikarenakan *throughput* bergantung pada jumlah flit yang diterima per cycle nya sehingga tinggi atau rendahnya *packet injection rate* memberikan pengaruh terbesar. Jumlah flit yang dikirimkan sebagai akibat dari ukuran paket memberikan pengaruh kedua. Perubahan nilai dari *delay* sangat berpengaruh terhadap perubahan nilai ukuran paket dimana dengan ukuran *buffer* yang tetap dan *packet injection rate* yang tetap, semakin banyaknya flit yang dikirimkan akan meningkatkan kemungkinan untuk terjadinya Congestion yang dapat meningkatkan *delay*. Sedangkan ukuran jaringan menjadi parameter desain yang sangat mempengaruhi perubahan nilai dari daya yang digunakan hal ini dikarenakan daya yang digunakan pada jaringan bergantung pada banyak faktor seperti proses pada router dan waktu atau cycles yang diperlukan oleh flit untuk mencapai tujuan. Semakin besar ukuran jaringan akan menyebabkan semakin banyak router yang harus dilewati dan semakin lama cycles yang diperlukan untuk mencapai node tujuan.

Pada jaringan dengan flow control Ack, perubahan dari throughput sangat dipengaruhi oleh perubahan pada *packet injection rate*.

Tabel 4.17 Pengaruh Perubahan Parameter Desain dengan Ack / Nack *Flow Control*

Ack / Nack *Flow Control*				
Parameter Performansi	Parameter Desain			
	Ukuran Jaringan	*Packet injection rate*	Ukuran Paket	Ukuran *Buffer*
Throughput	Keempat	Pertama	Kedua	Ketiga
Delay	Kedua	Ketiga	Pertama	Keempat
Daya	Pertama	Ketiga	Kedua	Keempat

Hal ini dikarenakan *throughput* bergantung pada jumlah flit yang diterima per cycle nya sehingga tinggi atau rendahnya *packet injection rate* memberikan pengaruh terbesar. Jumlah flit yang dikirimkan sebagai akibat dari ukuran paket memberikan pengaruh kedua. Perubahan nilai dari *delay* sangat berpengaruh terhadap perubahan nilai ukuran paket dimana dengan ukuran *buffer* yang tetap dan *packet injection rate* yang tetap, semakin banyaknya flit yang dikirimkan akan meningkatkan kemungkinan untuk terjadinya Congestion yang dapat meningkatkan *delay*. Sedangkan ukuran jaringan menjadi parameter desain yang sangat mempengaruhi perubahan nilai dari daya yang digunakan hal ini dikarenakan daya yang digunakan pada jaringan bergantung pada banyak faktor seperti proses pada router dan waktu atau cycles yang diperlukan oleh flit untuk mencapai tujuan. Semakin besar ukuran jaringan akan menyebabkan semakin banyak router yang harus dilewati dan semakin lama cycles yang diperlukan untuk mencapai node tujuan.

Tabel 4.18 Pengaruh Perubahan Parameter Desain dengan Stall / Go *Flow Control*

Stall / Go *Flow Control*				
Parameter Performansi	Parameter Desain			
	Ukuran Jaringan	*Packet injection rate*	Ukuran Paket	Ukuran *Buffer*
Throughput	Ketiga	Kedua	Pertama	Keempat
Delay	Ketiga	Kedua	Pertama	Keempat
Daya	Pertama	Ketiga	Kedua	Keempat

Pada jaringan dengan flow control Stall, perubahan dari throughput sangat dipengaruhi oleh perubahan pada ukuran paket .Hal ini dikarenakan *throughput* bergantung pada jumlah flit yang diterima per cycle nya sehingga besar atua kecilnya ukuran dari paket yang dikirimkan memberikan pengaruh terbesar. Perubahan nilai dari *delay* sangat berpengaruh terhadap perubahan nilai ukuran paket dimana dengan ukuran *buffer* yang tetap dan *packet injection rate* yang tetap, semakin banyaknya flit yang dikirimkan akan meningkatkan kemungkinan untuk terjadinya Congestion yang dapat meningkatkan *delay*. Sedangkan ukuran jaringan menjadi parameter desain yang sangat mempengaruhi perubahan nilai dari daya yang digunakan hal ini dikarenakan daya yang digunakan pada jaringan bergantung pada banyak faktor seperti proses pada router dan waktu atau cycles yang diperlukan oleh flit untuk mencapai tujuan. Semakin besar ukuran jaringan akan menyebabkan semakin banyak router yang harus dilewati dan semakin lama cycles yang diperlukan untuk mencapai node tujuan.

Tabel 4.19 Pengaruh Perubahan Parameter Desain dengan DyML *Flow Control*

Dynamic MultiLevel *Flow Control*				
Parameter Performansi	Parameter Desain			
	Ukuran Jaringan	*Packet injection rate*	Ukuran Paket	Ukuran *Buffer*
Throughput	Ketiga	Pertama	Kedua	Keempat
Delay	Kedua	Ketiga	Pertama	Keempat
Daya	Pertama	Ketiga	Kedua	Keempat

Pada jaringan flow control DyML, perubahan dari throughput sangat dipengaruhi oleh perubahan pada *packet injection rate* .Hal ini dikarenakan *throughput* bergantung pada jumlah flit yang diterima per cycle nya sehingga tinggi atau rendahnya *packet injection rate* memberikan pengaruh terbesar. Jumlah flit yang dikirimkan sebagai akibat dari ukuran paket memberikan pengaruh kedua. Perubahan nilai dari *delay* sangat berpengaruh terhadap perubahan nilai ukuran paket dimana dengan ukuran *buffer* yang tetap dan *packet injection rate* yang tetap, semakin banyaknya flit yang dikirimkan akan meningkatkan kemungkinan untuk terjadinya Congestion yang dapat meningkatkan *delay*. Sedangkan ukuran jaringan menjadi parameter desain yang sangat mempengaruhi perubahan nilai dari daya yang digunakan hal ini dikarenakan daya yang digunakan pada jaringan bergantung pada

banyak faktor seperti proses pada router dan waktu atau cycles yang diperlukan oleh flit untuk mencapai tujuan.

Tabel 4.20 menunjukkan karakteristik flow control setelah dilakukan perbandingan performansi jaringan.

Tabel 4.20 Karakteristik Setiap Flow Control

Flow Control	Kelebihan	Kekurangan
Stall / Go	• Tidak menyebabkan terjadinya saturasi throughput apabila packet injection rate dan ukuran paket diperbesar • Perubahan ukuran buffer dan ukuran jaringan tidak memberikan pengaruh pada delay dan konsumsi daya • Memberikan perbaikan nilai performansi terbaik untuk semua skenario	• Memberikan throughput yang tetap untuk perubahan ukuran jaringan dan perubahan ukuran buffer • Peningkatan nilai PIR memberikan peningkatan delay pada jaringan
Ack/ Nack	• Perubahan PIR dan ukuran buffer menghasilkan peningkatan throughput • Perubahan PIR menyebabkan penurunan pada delay	• Menyebabkan saturasi throughput apabila ukuran paket diperbesar
DyML	• Memberikan peningkatan throughput apabila buffer ditingkatkan • Memberikan penurunan konsumsi daya apabila ukuran jaringan semakiin diperbesar	• Menyebabkan saturasi throughput apabila ukuran paket diperbesar • Memberikan nilai delay yang lebih besar untuk perubahan PIR dibanding flow control lainnya

BAB 5
KESIMPULAN

Setelah dilakukan pengambilan data dan melakukan Analisis terhadap data hasil penelitian maka dapat ditarik kesimpulan terkait Analisis ketiga metode *flow control* yang digunakan pada *Network on Chip*. Selain itu disertakan pula kendala dan saran terhadap penelitian ini yang dapat digunakan untuk pengembangan dan penelitian lebih lanjut diwaktu yang akan datang.

5.1 Kesimpulan

Kesimpulan yang dapat diambil dari penelitian ini setelah dilakukan pengukuran dan analisis data dari hasil ketiga metode *flow control* yang diperoleh adalah:

1. *Flow Control* Stall / Go, Ack / Nack dan DyML memberikan pengaruh pada jaringan yang mengalami saturasi
2. Dengan kombinasi parameter desain *Packet injection rate* dan ukuran *buffer*, *flow control* Stall / Go, Ack / Nack dan DyML tidak menimbulkan saturasi hingga *packet injection rate* yang digunakan sebesar 0.3
3. Dengan kombinasi parameter desain Ukuran Paket dan *Packet injection rate*, *flow control* Stall / Go tidak menimbulkan saturasi hingga ukuran paket yang digunakan sebesar 20 – 22 flit.
4. Untuk dapat meningkatkan *throughput*, maka *Packet injection rate*, Ukuran Paket, Ukuran *Buffer* dan Ukuran Jaringan harus diperbesar,
5. Untuk dapat menurunkan *delay*, maka *Packet injection rate*, Ukuran Paket dan Ukuran Jaringan harus diperkecil dengan Ukuran *Buffer* yang diperbesar
6. Untuk dapat mengurangi penggunaan daya, maka *Packet injection rate*, Ukuran Paket, Ukuran *Buffer* dan Ukuran Jaringan harus diperkecil
7. *Flow Control* Stall / Go memberikan hasil terbaik dalam peningkatan *throughput* dengan peningkatan *throughput* rata – rata terhadap *flow control* default sebesar 21.08 % untuk ukuran jaringan yang berubah – ubah, 65.33 % untuk *packet injection rate* yang berubah – ubah, 151 % untuk ukuran paket

yang berubah – ubah dan 13.37 % untuk ukuran *buffer* yang berubah – ubah.

8. *Flow Control* Stall / Go memberikan hasil terbaik dalam penurunan *delay* dengan penurunan *delay* rata – rata terhadap *flow control* default sebesar 407.85 cycles untuk ukuran jaringan yang berubah – ubah, 606.03 cycles untuk *packet injection rate* yang berubah – ubah, 1631.95 cycles untuk ukuran paket yang berubah – ubah dan 322.59 cycles untuk ukuran *buffer* yang berubah – ubah.
9. *Flow Control* Stall / Go memberikan hasil terbaik dalam peningkatan *throughput* dengan peningkatan *throughput* rata – rata terhadap *flow control* default sebesar 68.67 % untuk ukuran jaringan yang berubah – ubah, 61.33 % untuk *packet injection rate* yang berubah – ubah, 49.93 % untuk ukuran paket yang berubah – ubah dan 68.22 % untuk ukuran *buffer* yang berubah – ubah.

5.2 Saran

Saran untuk pengembangan analisis performa *flow control* pada *Network on Chip* yaitu :

1. Melakukan perbandingan untuk kombinasi parameter desain yang lain untuk dapat mengetahui pengaruhnya terhadap kondisi saturasi jaringan
2. Melakukan penelitian lebih lanjut untuk pengaruh parameter desain lainnya terhadap performa jaringan seperti *throughput*, *delay*, jumlah paket yang diterima, dan daya yang digunakan
3. Melakukan perancangan dan implementasi desain pada perangkat hardware
4. Melakukan perancangan dan implementasi *flow control* pada perangkat hardware

DAFTAR PUSTAKA

[1] J. Howard, et al., "A 48 – *Core* IA-32 Processor in 45 nm CMOS Using On – Die Message – Passing and DVFS for Performance and Power Scaling," IEEE Journal of Solid State Circuits, vol. 46, no.1, pp. 173-183 jan 2011

[2] J. Kumar, A.Kanta, T. N. Kamal, K.K. Mahaptra, "Performance Evaluation of Different Routing Algorithms in Newtork on Chip", IEEE Asia Pacific Conference on Post Graduate Research in Microelectronics and Electronics, 2013

[3] R. Marculescu, U.Y. Orgras, L. –S. Peh, N. E. Jerger, Y. Hoskote, "Outstanding Research Problems in NoC Design : System, Microarchitecture," IEEE Trans. Computer – aided Design of Integrated Circuits System, Vol.28, no.1, pp. 3-21 Jan, 2009

[4] W. C. Tsai, Y. C. Lan, S. J. Chen, Y. Hen Hu, "DyML : Dynamic Multi Level *Flow Control* for *Network on Chip*", IEEE 2010

[5] Williams James Dally and Brian Towles, "Principles and Practices of Interconnection Networks", Morgan Kaufmaan, 2004

[6] V. V. Nimbalkar, K.Varghese, "In – Channel *flow control* scheme for *Network on Chip*", IEEE 13th Euromicro Conference on Digital System Design Architectures, Methods and Tools, 2010

[7] J. Hu, R. Marculescu, "DyAD – Smart Routing for Networks on Chip" Proceedings 41st Design Automation Conference, 2004

[8] W. Zhang, L. Hou, J.Wang, S.Geng, W. Wu, "Comparison Research between XY and Odd – Even routing algorithm of a 2-Dimension 3x3 Mesh Topology *Network on Chip*", IEEE Global Congress on Inteligent Systems, 2009

[9] XIAO Canwen, Z. Minxuan, D. Yong, Z. Zhitong, "Dimensional Bubble *Flow control* and Fully Adaptive Routing in the 2-D Mesh Newtork on Chip ", IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008

[10] A. Pulini, F. Angiolini, D. Bertozzi, L. Benini, "Fault Tolerance Overhead in Networks on Chip *Flow Control* Schemes", ACM, SBCCI, brazil, September 4-7, 2005

[11] Y.C. Lan, M.C. Chen, A.P. Su, Y.H. Hu and S.J. Chen, "Fluidity Concept for NoC : A *Congestion* Avoidance and Relief Routing Scheme," IEEE 21st International SoC Conference pp 65-70, 2008

[12] M. Bakhouya, A. Chariete, J.Gaber, M. Wack, S. Niar, E. COatanea "Performance Evaluation of a Flow – Control Algorithm for *Network on Chip* ", IEEE 2012
[13] I. Pratomo, "Adaptive NoC for Reconfigurable SoC"
[14] C. Nicopoulos, V. Narayanan, C.R.Das "Network on Chi Architectures : A holistic design exploration", Springer ,2009
[15] F. Gebali, H. Elmiligi ,M.W. El-Kharashi,"Networks on Chips : Theory and Practice ", CRC Press, 2009

LAMPIRAN

Listing Program untuk STALL/GO *Flow Control*

Berikut ini adalah beberapa script yang dirubah dalam beberapa file source untuk menerapkan STALL/GO *flow control*:

NoximProcessingElement.h

```
sc_in < NoximFlit > flit_rx;
sc_in < int > req_rx;
sc_out < int > stall_go_rx;

sc_out < NoximFlit > flit_tx;
sc_out < int > req_tx;
sc_in < int > stall_go_tx;
```

NoximProcessingElement.cpp

```
void NoximProcessingElement::rxProcess()
{
    if (reset.read())
    {
        stall_go_rx.write(1);
        current_level_rx=0;
    }
    else
    {
        if(req_rx.read()== 1 - current_level_rx)
        {
            NoximFlit flit_tmp = flit_rx.read();
            if(NoximGlobalParams::verbose_mode> VERBOSE_OFF)
            {
                cout<<sc_simulation_time()<<":
                ProcessingELement["<< local_id << "]
                RECEIVING " << flit_tmp <<endl;
            }
            current_level_rx=1- current_level_rx;
        }
        stall_go_rx.write(1);
    }
}

void NoximProcessingElement::txProcess()
{
    if(reset.read())
    {
```

```
        req_tx.write(0);
        current_level_tx=0;
        transmittedAtPreviousCycle = false;
    }
    else
    {
        NoximPacket packet;

        if(canShot(packet))
        {
            packet_queue.push(packet);
            transmittedAtPreviousCycle = true;
        }
        else
            transmittedAtPreviousCycle = false;
    }

    if (stall_go_tx.read() == 1)
    {
        if (!packet_queue.empty())
        {
            NoximFlit flit = nextFlit();
            if(NoximGlobalParams::verbose_mode>VERBOSE_OFF)
            {
                cout << sc_time_stamp().to_double()/ 1000
                     << ": ProcessingElement[" << local_id
                     << "] SENDING " << flit <<endl;
            }

            flit_tx -> write(flit);
            current_level_tx = 1 - current_level_tx;
            req_tx.write(current_level_tx);
        }
    }
}
```

NoximRouter.h

```
// I/O Ports
sc_in <NoximFlit> flit_rx[DIRECTIONS + 1];
sc_in <int> req_rx[DIRECTIONS + 1];
sc_out <int> stall_go_rx[DIRECTIONS + 1];

sc_out <NoximFlit> flit_tx[DIRECTIONS + 1];
sc_out <int> req_tx[DIRECTIONS + 1];
sc_in <int> stall_go_tx[DIRECTIONS + 1];

// Registers
int buffer_size;
```

```
int f_off;
int f_on;
```

NoximRouter.cpp

```
void NoximRouter::rxProcess()
{
    if(reset.read())
    {
        for(int i = 0; i < DIRECTIONS + 1; i++)
        {
            stall_go_rx[i].write(1);
            current_level_rx[i] = 0;
        }
        reservation_table.clear();
        routed_flits=0;
        local_drained=0;
    }
    else
    {
        for(int i = 0; i < DIRECTIONS + 1; i++)
        {
            if ((req_rx[i].read()==1 - current_level_rx[i])
                 && !buffer[i].IsFull())
            {
                NoximFlit received_flit = flit_rx[i].read();

               if(NoximGlobalParams::verbose_mode
                  >VERBOSE_OFF)
                {
                    cout << sc_time_stamp().to_double()/
                    1000 << ": Router[" << local_id << "],
                   Input["<< i << "], Received flit: "<<
                   received_flit<<endl;
                }

                buffer[i].Push(received_flit);
                current_level_rx[i]=1 - current_level_rx[i];

                stats.power.Buffering();

                if (received_flit.src_id == local_id)
                    stats.power.EndToEnd();
            }
        }
    }
    stats.power.Leakage();
}
```

```
void NoximRouter::txProcess()
{
    if (reset.read())
    {
        for (int i = 0; i < DIRECTIONS + 1; i++)
        {
            req_tx[i].write(0);
            current_level_tx[i] = 0;
        }
    }
    else
    {
        for (int i=0; i < DIRECTIONS + 1; i++)
        {
            if (!buffer[i].IsEmpty())
            {
                NoximFlit flit = buffer [i].Front();

                NoximRouteData route_data;
                    route_data.current_id = local_id;
                    route_data.src_id = flit.src_id;
                    route_data.dst_id = flit.dst_id;
                    route_data.dir_in = i;

                int o = reservation_table.getOutputPort(i);
                if (o != NOT_RESERVED)
                {
                    if((stall_go_tx[o].read()==1)
                       && (o == DIRECTION_LOCAL))
                    {
                        buffer_size = buffer[i].Size();
                        f_off = round(0.8*(buffer[i].
                                GetMaxBufferSize()));
                        f_on = round(0.2*(buffer[i].
                                GetMaxBufferSize()));

                        if(buffer_size >= f_off)
                        {
                            stall_go_rx[i].write(0);
                            while(buffer_size > f_on)
                            {
                                flit_tx[o].write(flit);
                                current_level_tx[o]=
                                   current_level_tx[o]+ 1;

                              req_tx[o].write
                                    (current_level_tx[o]);
                                buffer[i].Pop();
                            }
```

```
                    stall_go_rx[i].write(1);
                }
                else
                {
                    flit_tx[o].write(flit);
                    current_level_tx[o] =
                      current_level_tx[o] + 1;

                     req_tx[o].write
                       (current_level_tx[o]);
                    buffer[i].Pop();
                }

                if (NoximGlobalParams::
                       low_power_link_strategy)
                {
                    if(flit.flit_type
                       == FLIT_TYPE_HEAD ||
                    flit.use_low_voltage_path
                       == false)
                        stats.power.Link(false);
                    else
                        stats.power.Link(true);
                }
                else
                    stats.power.Link(false);

                if (flit.dst_id == local_id)
                    stats.power.EndToEnd();

                if (flit.flit_type
                    == FLIT_TYPE_TAIL)
                    reservation_table.release(o);

                if ( o == DIRECTION_LOCAL)
                {
                    stats.receivedFlit(
                     sc_time_stamp().to_double()
                     /1000,flit);

                    if (NoximGlobalParams::
                        max_volume_to_be_drained)
                    {
                        if(drained_volume >=
                         NoximGlobalParams
                        ::max_volume_to_be_drained)
                            sc_stop();
                        else
                        {
```

```
                                drained_volume++;
                                local_drained++;
                            }
                        }
                    }
                    else if (i != DIRECTION_LOCAL)
                    {
                        routed_flits++;
                    }
                }
                else if ((stall_go_tx[o].read() == 1))
                {
                    flit_tx[o].write(flit);
                    current_level_tx[o] =
                       current_level_tx[o] + 1;
                    req_tx[o].write
                        (current_level_tx[o]);
                    buffer[i].Pop();

                    if (NoximGlobalParams::
                         low_power_link_strategy)
                    {
                        if(flit.flit_type ==
                        FLIT_TYPE_HEAD ||
                        flit.use_low_voltage_path
                        == false)
                            stats.power.Link(false);
                        else
                            stats.power.Link(true);
                    }
                    else
                        stats.power.Link(false);

                    if (flit.dst_id == local_id)
                        stats.power.EndToEnd();

                    if (flit.flit_type
                         == FLIT_TYPE_TAIL)
                        reservation_table.release(o);

                    if ( o == DIRECTION_LOCAL)
                    {
                        stats.receivedFlit
                        (sc_time_stamp().to_double()
                         /1000,flit);

                        if (NoximGlobalParams
                            ::max_volume_to_be_drained)
                        {
```

```
                                if(drained_volume >=
                                NoximGlobalParams::
                                max_volume_to_be_drained)
                                    sc_stop();
                                else
                                {
                                    drained_volume++;
                                    local_drained++;
                                }
                            }
                        }
                        else if (i != DIRECTION_LOCAL)
                        {
                            routed_flits++;
                        }
                    }
                    else if (stall_go_tx[o].read() == 0)
                    {
                        stall_go_rx[i].write(0);
                    }
                }
            }
        }
    }
    stats.power.Leakage();
}
```

NoximNoC.cpp

```
t[i][j]->req_rx[DIRECTION_NORTH] (req_to_south[i][j]);
t[i][j]->flit_rx[DIRECTION_NORTH] (flit_to_south[i][j]);
t[i][j]->stall_go_rx[DIRECTION_NORTH]
        (stall_go_to_north[i][j]);

t[i][j]->req_rx[DIRECTION_EAST] (req_to_west[i + 1][j]);
t[i][j]->flit_rx[DIRECTION_EAST] (flit_to_west[i + 1][j]);
t[i][j]->stall_go_rx[DIRECTION_EAST] (stall_go_to_east[i +
        1][j]);

t[i][j]->req_rx[DIRECTION_SOUTH] (req_to_north[i][j + 1]);
t[i][j]->flit_rx[DIRECTION_SOUTH] (flit_to_north[i][j + 1]);
t[i][j]->stall_go_rx[DIRECTION_SOUTH]
       (stall_go_to_south[i][j + 1]);

t[i][j]->req_rx[DIRECTION_WEST] (req_to_east[i][j]);
t[i][j]->flit_rx[DIRECTION_WEST] (flit_to_east[i][j]);
t[i][j]->stall_go_rx[DIRECTION_WEST]
        (stall_go_to_west[i][j]);
```

```
// Map Tx signals
t[i][j]->req_tx[DIRECTION_NORTH] (req_to_north[i][j]);
t[i][j]->flit_tx[DIRECTION_NORTH] (flit_to_north[i][j]);
t[i][j]->stall_go_tx[DIRECTION_NORTH]
        (stall_go_to_south[i][j]);

t[i][j]->req_tx[DIRECTION_EAST] (req_to_east[i + 1][j]);
t[i][j]->flit_tx[DIRECTION_EAST] (flit_to_east[i + 1][j]);
t[i][j]->stall_go_tx[DIRECTION_EAST] (stall_go_to_west[i +
        1][j]);

t[i][j]->req_tx[DIRECTION_SOUTH] (req_to_south[i][j + 1]);
t[i][j]->flit_tx[DIRECTION_SOUTH] (flit_to_south[i][j + 1]);
t[i][j]->stall_go_tx[DIRECTION_SOUTH]
       (stall_go_to_north[i][j + 1]);

t[i][j]->req_tx[DIRECTION_WEST] (req_to_west[i][j]);
t[i][j]->flit_tx[DIRECTION_WEST] (flit_to_west[i][j]);
t[i][j]->stall_go_tx[DIRECTION_WEST]
        (stall_go_to_east[i][j]);
```

NoximNoC.h

```
// Signals
sc_signal <int> req_to_east
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_west
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_south
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_north
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];

sc_signal <int> stall_go_to_east
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> stall_go_to_west
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> stall_go_to_south
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> stall_go_to_north
               [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
```

NoximTile.h

```
 // I/O Ports
sc_in_clk clock;
sc_in <bool> reset;

sc_in <NoximFlit> flit_rx[DIRECTIONS];
```

```
sc_in <int> req_rx[DIRECTIONS];
sc_out <int> stall_go_rx[DIRECTIONS];

sc_out <NoximFlit> flit_tx[DIRECTIONS];
sc_out <int> req_tx[DIRECTIONS];
sc_in <int> stall_go_tx[DIRECTIONS];

sc_out <int> free_slots[DIRECTIONS];
sc_in <int> free_slots_neighbor[DIRECTIONS];

// NoP related I/O
sc_out < NoximNoP_data > NoP_data_out[DIRECTIONS];
sc_in < NoximNoP_data > NoP_data_in[DIRECTIONS];

// Signals
sc_signal <NoximFlit> flit_rx_local;
sc_signal <int> req_rx_local;
sc_signal <int> stall_go_rx_local;

sc_signal <NoximFlit> flit_tx_local;
sc_signal <int> req_tx_local;
sc_signal <int> stall_go_tx_local;
```

Listing Program ACK/NACK *Flow Control*

Berikut ini adalah beberapa script yang digunakan dalam beberapa file source untuk menerapkan ACK/NACK *flow control*:

NoximProcessingElement.h

```
// I/O Ports
sc_in_clk clock;
sc_in < bool > reset;

sc_in < NoximFlit > flit_rx;
sc_in < int > req_rx;
sc_out < int > ack_rx;

sc_out < NoximFlit > flit_tx;
sc_out < int > req_tx;
sc_in < int > ack_tx;
```

NoximProcessingElement.cpp

```
void NoximProcessingElement::rxProcess()
{
    if(reset.read())
    {
        ack_rx.write(1);
```

```
        sequences_rx = 0;
    }
    else
    {
        if(req_rx.read() == 1 - sequences_rx)
        {
            NoximFlit flit_tmp = flit_rx.read();
            if (NoximGlobalParams::verbose_mode >
                 VERBOSE_OFF)
            {
                cout << sc_simulation_time() <<" :
                     ProcessingELement["
                     << local_id << "] RECEIVING " <<
                     flit_tmp <<endl;
            }
            sequences_rx = 1 - sequences_rx;
        }
        ack_rx.write(1);
    }
}

void NoximProcessingElement::txProcess()
{
    if(reset.read())
    {
        req_tx.write(0);
        sequences_tx=0;
        transmittedAtPreviousCycle = false;
    }
    else
    {
        NoximPacket packet;

        if(canShot(packet))
        {
            packet_queue.push(packet);
            transmittedAtPreviousCycle = true;
        }
        else
            transmittedAtPreviousCycle = false;

        if (ack_tx.read() > 0)
        {

            if(!packet_queue.empty())
            {
                NoximFlit flit = nextFlit();

                }
```

```
                flit_tx -> write(flit);
                sequences_tx = 1 - sequences_tx;
                req_tx.write(sequences_tx);
            }
            else if(ack_tx.read() == 2)
            {
                transmittedAtPreviousCycle = true;
                transmittedAtPreviousCycle = false;
            }
        }

    }
}
```

NoximRouter.h

```
// I/O Ports
sc_in_clk clock;
sc_in <bool> reset;

sc_in <NoximFlit> flit_rx[DIRECTIONS + 1];
sc_in <int> req_rx[DIRECTIONS + 1];
sc_out <int> ack_rx[DIRECTIONS + 1];

sc_out <NoximFlit> flit_tx[DIRECTIONS + 1];
sc_out <int> req_tx[DIRECTIONS + 1];
sc_in <int> ack_tx[DIRECTIONS + 1];
```

NoximRouter.cpp

```
void NoximRouter::rxProcess()
{
    if(reset.read())
    {
        for(int i=0; i < DIRECTIONS + 1; i++)
        {
            ack_rx[i].write(1);
            sequences_rx[i] = 0;
        }
        reservation_table.clear();
        routed_flits=0;
        local_drained=0;
    }
    else
    {
        for(int i=0; i < DIRECTIONS + 1; i++)
        {
            if(req_rx[i].read() == 1 - sequences_rx[i])
```

```
            {
                NoximFlit received_flit = flit_rx[i].read();
                NoximRouteData route_data;
                route_data.current_id = local_id;
                route_data.dst_id = received_flit.dst_id;

                if(route_data.current_id ==
                   route_data.dst_id)
                {
                    if(!buffer[i].IsFull())
                    {

                        buffer[i].Push(received_flit);
                        sequences_rx[i] = 1 –
                                          sequences_rx[i];

                        stats.power.Buffering();

                        if(received_flit.src_id == local_id)
                            stats.power.EndToEnd();

                        ack_rx[i].write(1);
                    }
                    else if(buffer[i].IsFull())
                    {
                        ack_rx[i].write(2);
                    }
                }
                else
                {
                    if(!buffer[i].IsFull())
                    {

                        buffer[i].Push(received_flit);
                        sequences_rx[i] = 1 –
                                          sequences_rx[i];

                        stats.power.Buffering();

                        if(received_flit.src_id == local_id)
                            stats.power.EndToEnd();
                    }
                }
            }
        }
    }
    stats.power.Leakage();
}
```

```
void NoximRouter::txProcess()
{
    if(reset.read())
    {
        for(int i=0; i < DIRECTIONS + 1;i++)
        {
            req_tx[i].write(0);
            sequences_tx[i] = 0;
        }
    }
    else
    {
for (int i = 0; i < DIRECTIONS + 1; i++)
        {
            if(!buffer[i].IsEmpty())
            {
                NoximFlit flit = buffer[i].Front();

                int o = reservation_table.getOutputPort(i);
                if(o != NOT_RESERVED)
                {
                    if (ack_tx[o].read() == 2)
                    {

                        flit_tx[o].write(flit);
                        sequences_tx[o]= 1 –
                                              sequences_tx[o];
                        req_tx[o].write(sequences_tx[o]);
                        buffer[i].Pop();

                        if (NoximGlobalParams::
                                    low_power_link_strategy)
                        {
                            if(flit.flit_type ==
                               FLIT_TYPE_HEAD ||
                               flit.use_low_voltage_path
                               == false)
                                stats.power.Link(false);
                            else
                                stats.power.Link(true);
                        }
                        else
                            stats.power.Link(false);

                        if(flit.dst_id == local_id)
                            stats.power.EndToEnd();

                        if(flit.flit_type == FLIT_TYPE_TAIL)
                            reservation_table.release(o);
```

```
                    if (o == DIRECTION_LOCAL)
                    {
                        stats.receivedFlit
                       (sc_time_stamp().to_double()
                        / 1000, flit);
                        if (NoximGlobalParams::
                            max_volume_to_be_drained)
                        {
                            if (drained_volume >=
                                NoximGlobalParams::
                               max_volume_to_be_drained)
                                sc_stop();
                            else
                            {
                                drained_volume++;
                                local_drained++;
                            }
                        }
                    }

                    else if(i != DIRECTION_LOCAL)
                    {
                        routed_flits++;
                    }
                }
                else if (ack_tx[o].read() == 1)
                {
                    acktx=ack_tx[o].read();

                    flit_tx[o].write(flit);
                    sequences_tx[o]=1 - sequences_tx[o];
                    req_tx[o].write(sequences_tx[o]);
                    buffer[i].Pop();

                    if (NoximGlobalParams::
                          low_power_link_strategy)
                    {
                        if(flit.flit_type ==
                          FLIT_TYPE_HEAD
                          || flit.use_low_voltage_path
                          == false)
                            stats.power.Link(false);
                        else
                            stats.power.Link(true);
                    }
                    else
                        stats.power.Link(false);
```

```
                        if(flit.dst_id == local_id)
                            stats.power.EndToEnd();

                        if(flit.flit_type == FLIT_TYPE_TAIL)
                            reservation_table.release(o);

                        if (o == DIRECTION_LOCAL)
                        {
                            stats.receivedFlit
                           (sc_time_stamp().to_double()
                            / 1000, flit);
                            if (NoximGlobalParams::
                              max_volume_to_be_drained)
                            {
                                if (drained_volume >=
                                   NoximGlobalParams::
                                    max_volume_to_be_drained)
                                    sc_stop();
                                else
                                {
                                    drained_volume++;
                                    local_drained++;
                                }
                            }
                        }

                        else if( i != DIRECTION_LOCAL)
                        {
                            routed_flits++;
                        }
                    }

                }
            }
        }
    }
    stats.power.Leakage();
}
```

NoximNoC.cpp

```
t[i][j]->req_rx[DIRECTION_NORTH] (req_to_south[i][j]);
t[i][j]->flit_rx[DIRECTION_NORTH] (flit_to_south[i][j]);
t[i][j]->ack_rx[DIRECTION_NORTH]
        (ack_to_north[i][j]);

t[i][j]->req_rx[DIRECTION_EAST] (req_to_west[i + 1][j]);
t[i][j]->flit_rx[DIRECTION_EAST] (flit_to_west[i + 1][j]);
t[i][j]->ack_rx[DIRECTION_EAST] (ack_to_east[i +
```

```
        1][j]);

t[i][j]->req_rx[DIRECTION_SOUTH] (req_to_north[i][j + 1]);
t[i][j]->flit_rx[DIRECTION_SOUTH] (flit_to_north[i][j + 1]);
t[i][j]->ack_rx[DIRECTION_SOUTH]
       (ack_to_south[i][j + 1]);

t[i][j]->req_rx[DIRECTION_WEST] (req_to_east[i][j]);
t[i][j]->flit_rx[DIRECTION_WEST] (flit_to_east[i][j]);
t[i][j]->ack_rx[DIRECTION_WEST]
        (ack_to_west[i][j]);

// Map Tx signals
t[i][j]->req_tx[DIRECTION_NORTH] (req_to_north[i][j]);
t[i][j]->flit_tx[DIRECTION_NORTH] (flit_to_north[i][j]);
t[i][j]->ack_tx[DIRECTION_NORTH]
        (ack_to_south[i][j]);

t[i][j]->req_tx[DIRECTION_EAST] (req_to_east[i + 1][j]);
t[i][j]->flit_tx[DIRECTION_EAST] (flit_to_east[i + 1][j]);
t[i][j]->ack_tx[DIRECTION_EAST] (ack_to_west[i +
        1][j]);

t[i][j]->req_tx[DIRECTION_SOUTH] (req_to_south[i][j + 1]);
t[i][j]->flit_tx[DIRECTION_SOUTH] (flit_to_south[i][j + 1]);
t[i][j]->ack_tx[DIRECTION_SOUTH]
       (ack_to_north[i][j + 1]);

t[i][j]->req_tx[DIRECTION_WEST] (req_to_west[i][j]);
t[i][j]->flit_tx[DIRECTION_WEST] (flit_to_west[i][j]);
t[i][j]->ack_tx[DIRECTION_WEST]
        (ack_to_east[i][j]);
```

NoximNoC.h

```
// Signals
sc_signal <int> req_to_east
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_west
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_south
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_north
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];

sc_signal <int> ack_to_east
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> ack_to_west
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
```

```
sc_signal <int> ack_to_south
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> ack_to_north
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
```

NoximTile.h

```
 // I/O Ports
sc_in_clk clock;
sc_in <bool> reset;

sc_in <NoximFlit> flit_rx[DIRECTIONS];
sc_in <int> req_rx[DIRECTIONS];
sc_out <int> ack_rx[DIRECTIONS];

sc_out <NoximFlit> flit_tx[DIRECTIONS];
sc_out <int> req_tx[DIRECTIONS];
sc_in <int> ack_tx[DIRECTIONS];

sc_out <int> free_slots[DIRECTIONS];
sc_in <int> free_slots_neighbor[DIRECTIONS];

// NoP related I/O
sc_out < NoximNoP_data > NoP_data_out[DIRECTIONS];
sc_in < NoximNoP_data > NoP_data_in[DIRECTIONS];

// Signals
sc_signal <NoximFlit> flit_rx_local;
sc_signal <int> req_rx_local;
sc_signal <int> ack_rx_local;

sc_signal <NoximFlit> flit_tx_local;
sc_signal <int> req_tx_local;
sc_signal <int> ack_tx_local;
```

Listing Program DyML *Flow Control*

Berikut ini adalah beberapa script yang digunakan dalam beberapa file source untuk menerapkan DyML *flow control*:

NoximProcessingElement.h

```
sc_in < NoximFlit > flit_rx;
sc_in < int > req_rx;
sc_out < int > stall_go_rx;

sc_out < NoximFlit > flit_tx;
sc_out < int > req_tx;
```

```
sc_in < int > stall_go_tx;
```

NoximProcessingElement.cpp

```
void NoximProcessingElement::rxProcess()
{
    if (reset.read())
    {
        stall_go_rx.write(1);
        current_level_rx=0;
    }
    else
    {
        if(req_rx.read() == 1 - current_level_rx)
        {
            NoximFlit flit_tmp = flit_rx.read();
            current_level_rx = 1 - current_level_rx;
        }
        stall_go_rx.write(1);
    }
}
```

NoximRouter.h

```
sc_in <NoximFlit> flit_rx[DIRECTIONS + 1];
sc_in <int> req_rx[DIRECTIONS + 1];
sc_out <int> stall_go_rx[DIRECTIONS + 1];

sc_out <NoximFlit> flit_tx[DIRECTIONS + 1];
sc_out <int> req_tx[DIRECTIONS + 1];
sc_in <int> stall_go_tx[DIRECTIONS + 1];
```

NoximRouter.cpp

```
void NoximRouter::rxProcess()
{
    if (reset.read())
    {
        for (int i = 0; i < DIRECTIONS + 1; i++)
        {
            current_level_rx[i] = 0;
        }
        reservation_table.clear();
        routed_flits = 0;
        local_drained = 0;
    }
    else
    {
        for(int i = 0; i < DIRECTIONS + 1; i++)
```

```
        {
            if ((req_rx[i].read() == 1 –
               current_level_rx[i]) && !buffer[i].IsFull())
            {
                NoximFlit received_flit = flit_rx[i].read();


                buffer[i].Push(received_flit);
                current_level_rx[i] = 1 –
                                        current_level_rx[i];

                stats.power.Buffering();

                if (received_flit.src_id == local_id)
                    stats.power.EndToEnd();
            }
        }
    }
    stats.power.Leakage();
}

void NoximRouter::txProcess()
{
    if (reset.read())
    {
        for(int i = 0; i < DIRECTIONS + 1; i++)
        {
            stall_go_rx[i].write(1);
            req_tx[i].write(0);
            current_level_tx[i] = 0;
            Bfluid[i] = 0;
            buffer_pop_counter[i] = 0;
            PTO[i]=1;
        }
    }
    else
    {
for (int i=0; i < DIRECTIONS + 1; i++)
        {
            range_time = sc_time_stamp().to_double()/1000;
            periodic_check_interval =
       0.01*(NoximGlobalParams::simulation_time);

            buffer_size = buffer[i].Size();
            buffer_pop_req = round(0.2*(buffer_size));

           max_buffer_size_buffer=
                              buffer[i].GetMaxBufferSize();
```

```
f_on = round(0.5*(buffer[i].
                              GetMaxBufferSize()));
f_off = round(0.9*(buffer[i].
                              GetMaxBufferSize()));

Bfill[i]=getBfill
           (buffer_size,max_buffer_size_buffer);

if((range_time % periodic_check_interval) == 0)
{
    PTO[i]=getPTO
         (buffer_pop_counter[i],buffer_pop_req);

    Bfill[i]=getBfill
           (buffer_size,max_buffer_size_buffer);

    Bfluid[i]=getBfluid
                (PTO[i],Bfluid[i],buffer_size);

    buffer_pop_counter[i] = 0;
}
if(!buffer[i].IsEmpty())
{
    NoximFlit flit = buffer[i].Front();

    NoximRouteData route_data;
    route_data.current_id = local_id;
    route_data.dst_id = flit.dst_id;

    int o = reservation_table.getOutputPort(i);
    if (o != NOT_RESERVED)
    {
        if(route_data.current_id ==
                              route_data.dst_id)
        {
            if(Bfill[i] == 0)
            {
                stall_go_rx[i].write(0);
                while (buffer_size > f_on)
                {
                    flit_tx[o].write(flit);
                    current_level_tx[o] =1 –
                             current_level_tx[o];
                    req_tx[o].write
                           (current_level_tx[o]);
                    buffer[i].Pop();
                    buffer_pop_counter[i] =
                       buffer_pop_counter[i] + 1;
                }
```

```
                        stall_go_rx[i].write(1);

                        if(NoximGlobalParams::
                                low_power_link_strategy)
                        {
                            if(flit.flit_type ==
                                FLIT_TYPE_HEAD
                            || flit.use_low_voltage_path
                             == false)
                                stats.power.Link(false);
                            else
                                stats.power.Link(true);
                        }
                        else
                            stats.power.Link(false);

                        if(flit.dst_id == local_id)
                            stats.power.EndToEnd();

                        if(flit.flit_type ==
                                            FLIT_TYPE_TAIL)
                            reservation_table.
                                                release(o);

                        if(o==DIRECTION_LOCAL)
                        {
                            stats.receivedFlit
                                 (sc_time_stamp().
                                 to_double()/1000,flit);

                            if(NoximGlobalParams::
                               max_volume_to_be_drained)
                            {
                                if(drained_volume >=
                              NoximGlobalParams
                              ::max_volume_to_be_drained)
                                    sc_stop();
                                else
                                {
                                    drained_volume++;
                                    local_drained++;
                                }
                            }
                        }
                        else if ( i != DIRECTION_LOCAL)
                        {
                            routed_flits++;
                        }
                    }
```

```
                    else if(stall_go_tx[o].read() == 1)
                    {

                        flit_tx[o].write(flit);
                        current_level_tx[o] =1 –
                                    current_level_tx[o];
                        req_tx[o].write
                                  (current_level_tx[o]);
                        buffer[i].Pop();
                        buffer_pop_counter[i] =
                               buffer_pop_counter[i] + 1;

                        if(NoximGlobalParams::
                                 low_power_link_strategy)
                        {
                            if(flit.flit_type ==
                                FLIT_TYPE_HEAD
                            || flit.use_low_voltage_path
                                 == false)
                                 stats.power.Link(false);
                            else
                                stats.power.Link(true);
                        }
                        else
                            stats.power.Link(false);

                        if(flit.dst_id == local_id)
                            stats.power.EndToEnd();

                        if(flit.flit_type ==
                                          FLIT_TYPE_TAIL)
                            reservation_table.
                             release(o);

                        if(o==DIRECTION_LOCAL)
                        {
                            stats.receivedFlit
                                 (sc_time_stamp().
                                 to_double()/1000,flit);

                            if(NoximGlobalParams::
                                max_volume_to_be_drained)
                            {
                                 if(drained_volume >=
                                    NoximGlobalParams
                                  ::max_volume_to_be_drained)
                                       sc_stop();
                                 else
                                 {
```

```
                                drained_volume++;
                                local_drained++;
                            }
                        }
                    }
                    else if ( i != DIRECTION_LOCAL)
                    {
                        routed_flits++;
                    }
                }
                else
                {
                    stall[i] = getStall
                               (Bfluid[i],Bfill[i]);

                    if(stall[i])
                    {
                        stall_go_rx[i].write(0);
                        while (buffer_size > f_on)
                        {
                            flit_tx[o].write(flit);
                            current_level_tx[o] =1 -
                                 current_level_tx[o];
                            req_tx[o].write
                              (current_level_tx[o]);
                            buffer[i].Pop();
                            buffer_pop_counter[i] =
                           buffer_pop_counter[i] + 1;
                        }
                        stall_go_rx[i].write(1);

                        if(NoximGlobalParams::
                             low_power_link_strategy)
                        {
                            if(flit.flit_type ==
                              FLIT_TYPE_HEAD
                            || flit.
                                 use_low_voltage_path
                             == false)
                                stats.power.
                                        Link(false);
                            else
                                stats.power.
                                         Link(true);
                        }
                        else
                            stats.power.Link(false);

                        if(flit.dst_id == local_id)
```

```
                                stats.power.EndToEnd();

                            if(flit.flit_type ==
                                        FLIT_TYPE_TAIL)
                                reservation_table.
                                            release(o);

                            if(o==DIRECTION_LOCAL)
                            {
                                stats.receivedFlit
                                  (sc_time_stamp().
                                  to_double()/1000,flit);

                                if(NoximGlobalParams::
                                max_volume_to_be_drained)
                                {
                                    if(drained_volume >=
                                      NoximGlobalParams::
                                max_volume_to_be_drained)
                                        sc_stop();
                                    else
                                    {
                                        drainedvolume++;
                                        local_drained++;
                                    }
                                }
                            }
                            else if (i !=
                                        DIRECTION_LOCAL)
                            {
                                routed_flits++;
                            }
                        }
                        else if(!stall)
                        {
                            stall_go_rx[i].write(1);
                        }
                    }

                }
                else
                {
                    if(stall_go_tx[o].read() == 1)
                    {

                        flit_tx[o].write(flit);
                        current_level_tx[o] =1 -
                                    current_level_tx[o];
                        req_tx[o].
```

```
                        write(current_level_tx[o]);
                    buffer[i].Pop();
                    buffer_pop_counter[i] =
                          buffer_pop_counter[i] + 1;

                    if(NoximGlobalParams::
                            low_power_link_strategy)
                    {
                        if(flit.flit_type ==
                          FLIT_TYPE_HEAD
                        || flit.use_low_voltage_path
                          == false)
                            stats.power.Link(false);
                        else
                            stats.power.Link(true);
                    }
                    else
                        stats.power.Link(false);

                    if(flit.dst_id == local_id)
                        stats.power.EndToEnd();

                    if(flit.flit_type ==
                                      FLIT_TYPE_TAIL)
                        reservation_table.
                                         release(o);

                    if(o==DIRECTION_LOCAL)
                    {
                        stats.receivedFlit
                                    (sc_time_stamp().
                              to_double()/1000,flit);

                        if(NoximGlobalParams::
                            max_volume_to_be_drained)
                        {
                            if(drained_volume >=
                                  NoximGlobalParams::
                            max_volume_to_be_drained)
                                sc_stop();
                            else
                            {
                                drained_volume++;
                                local_drained++;
                            }
                        }
                    }
                    else if ( i != DIRECTION_LOCAL)
                    {
```

```
                                routed_flits++;
                            }
                        }
                        else if(stall_go_tx[o].read() == 0)
                        {
                            stall_go_rx[i].write(0);
                        }
                        else
                        {
                            stall[i] = getStall
                                        (Bfluid[i],Bfill[i]);
                            if (stall[i])
                            {
                                stall_go_rx[i].write(0);
                            }
                            else
                            {
                                stall_go_rx[i].write(1);
                            }
                        }
                    }
                }
            }
        }
    }
    stats.power.Leakage();
}

int  NoximRouter::getPTO(int  buffer_pop_counter_values,  int
buffer_pop_req_values)
{

    int PTO_values;

    if(buffer_pop_counter_values >= buffer_pop_req_values)
    {
        PTO_values = 1;
    }
    else
    {
        PTO_values = 0;
    }

    return PTO_values;
}

int      NoximRouter::getBfluid(int      PTO_values,      int
Bfluid_values, int buffer_size_values)
{
```

```
    int Bfluid_values_new;

    if(PTO_values == 1 && Bfluid_values > 1)
    {
        Bfluid_values_new = Bfluid_values - 1;
    }
    else if(PTO_values != 1 && Bfluid_values > 0)
    {
        Bfluid_values_new = Bfluid_values +1;
    }
    else if(buffer_size_values == 0)
    {
        Bfluid_values_new = 0;
    }

    return Bfluid_values_new;
}

int    NoximRouter::getBfill    (int    buffer_size,    int
max_buffer_size)
{
    int Bfill_values;
    if (buffer_size == max_buffer_size)
    {
        Bfill_values = 0;
    }

    else if(buffer_size == (max_buffer_size) - 1)
    {
        Bfill_values = 1;
    }

    else if(buffer_size <= (0.5*(max_buffer_size)))
    {
        Bfill_values = 2;
    }

    else if(buffer_size <= (0.25*(max_buffer_size)))
    {
        Bfill_values = 3;
    }

    else if(buffer_size <=(0.125*(max_buffer_size)))
    {
        Bfill_values = 4;
    }

    return Bfill_values;
}
```

```
bool     NoximRouter::getStall(int     Bfluid_values,     int
Bfill_values)
{

    bool stall_values;

    if (Bfluid_values == 1 && Bfill_values ==1)
    {
        stall_values = true;
    }

    else if (Bfluid_values == 2 && Bfill_values == 2)
    {
        stall_values = true;
    }

    else if (Bfluid_values == 3 && Bfill_values == 3)
    {
        stall_values = true;
    }

    else if (Bfluid_values == 4 && Bfill_values == 4)
    {
        stall_values = true;
    }

    else if (Bfluid_values == 5)
    {
        stall_values = true;
    }

    else
    {
        stall_values = false;
    }

    return stall_values;
}
```

NoximNoC.cpp

```
t[i][j]->req_rx[DIRECTION_NORTH] (req_to_south[i][j]);
t[i][j]->flit_rx[DIRECTION_NORTH] (flit_to_south[i][j]);
t[i][j]->stall_go_rx[DIRECTION_NORTH]
         (stall_go_to_north[i][j]);

t[i][j]->req_rx[DIRECTION_EAST] (req_to_west[i + 1][j]);
t[i][j]->flit_rx[DIRECTION_EAST] (flit_to_west[i + 1][j]);
```

```
t[i][j]->stall_go_rx[DIRECTION_EAST] (stall_go_to_east[i +
        1][j]);

t[i][j]->req_rx[DIRECTION_SOUTH] (req_to_north[i][j + 1]);
t[i][j]->flit_rx[DIRECTION_SOUTH] (flit_to_north[i][j + 1]);
t[i][j]->stall_go_rx[DIRECTION_SOUTH]
       (stall_go_to_south[i][j + 1]);

t[i][j]->req_rx[DIRECTION_WEST] (req_to_east[i][j]);
t[i][j]->flit_rx[DIRECTION_WEST] (flit_to_east[i][j]);
t[i][j]->stall_go_rx[DIRECTION_WEST]
        (stall_go_to_west[i][j]);

// Map Tx signals
t[i][j]->req_tx[DIRECTION_NORTH] (req_to_north[i][j]);
t[i][j]->flit_tx[DIRECTION_NORTH] (flit_to_north[i][j]);
t[i][j]->stall_go_tx[DIRECTION_NORTH]
        (stall_go_to_south[i][j]);

t[i][j]->req_tx[DIRECTION_EAST] (req_to_east[i + 1][j]);
t[i][j]->flit_tx[DIRECTION_EAST] (flit_to_east[i + 1][j]);
t[i][j]->stall_go_tx[DIRECTION_EAST] (stall_go_to_west[i +
        1][j]);

t[i][j]->req_tx[DIRECTION_SOUTH] (req_to_south[i][j + 1]);
t[i][j]->flit_tx[DIRECTION_SOUTH] (flit_to_south[i][j + 1]);
t[i][j]->stall_go_tx[DIRECTION_SOUTH]
       (stall_go_to_north[i][j + 1]);

t[i][j]->req_tx[DIRECTION_WEST] (req_to_west[i][j]);
t[i][j]->flit_tx[DIRECTION_WEST] (flit_to_west[i][j]);
t[i][j]->stall_go_tx[DIRECTION_WEST]
        (stall_go_to_east[i][j]);
```

NoximNoC.h

```
// Signals
sc_signal <int> req_to_east
               [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_west
               [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_south
               [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> req_to_north
               [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];

sc_signal <int> stall_go_to_east
               [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> stall_go_to_west
```

```
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> stall_go_to_south
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
sc_signal <int> stall_go_to_north
                [MAX_STATIC_DIM + 1][MAX_STATIC_DIM + 1];
```

NoximTile.h

```
 // I/O Ports
sc_in_clk clock;
sc_in <bool> reset;

sc_in <NoximFlit> flit_rx[DIRECTIONS];
sc_in <int> req_rx[DIRECTIONS];
sc_out <int> stall_go_rx[DIRECTIONS];

sc_out <NoximFlit> flit_tx[DIRECTIONS];
sc_out <int> req_tx[DIRECTIONS];
sc_in <int> stall_go_tx[DIRECTIONS];

sc_out <int> free_slots[DIRECTIONS];
sc_in <int> free_slots_neighbor[DIRECTIONS];

// NoP related I/O
sc_out < NoximNoP_data > NoP_data_out[DIRECTIONS];
sc_in < NoximNoP_data > NoP_data_in[DIRECTIONS];

// Signals
sc_signal <NoximFlit> flit_rx_local;
sc_signal <int> req_rx_local;
sc_signal <int> stall_go_rx_local;

sc_signal <NoximFlit> flit_tx_local;
sc_signal <int> req_tx_local;
sc_signal <int> stall_go_tx_local;
```

BIOGRAFI PENULIS

Muhammad Hizrian Hizburrahman lahir di kota Jakarta pada tanggal 13 Mei 1994. Penulis mengenyam pendidikan dasar di Tangerang Selatan, SMPN 8 Tangerang Selatan dan SMAN 2 Tangerang Selatan sebelum akhirnya memuntuskan untuk melanjutkan pendidikan tinggi di Institut Teknologi Sepuluh Nopember (ITS) Surabaya. Penulis mengambil jurusan teknik elektro dan fokus pada bidang studi telekomunikasi multimedia. Selama aktif berkuliah di ITS, penulis mengikuti beberapa kepanitiaan mahasiswa. Pengalaman akademik dan aktivitas laboratorium sebagai asisten laboratorium Jaringan Telekomunikasi B.301 dan koordinator praktikum Komunikasi Data juga turut berkontribusi dalam mengembangkan kemampuan penulis selama berada di ITS.

DAFTAR GAMBAR

Analisis Unjuk Kerja *Flow Control* pada *Network on Chip* dalam Beberapa Kondisi Jaringan

M Hizrian Hizburrahman, Dr. Istas Pratomo, ST. MT, dan Ir. Djoko Suprajitno R, MT.
Jurusan Teknik Elektro, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: hizrian11@mhs.ee.its.ac.id , istaspra@ee.its.ac.id , djokosr@ee.its.ac.id

***Abstrak*—Semakin berkembangnya teknologi, ukuran dari perangkat keras yang digunakan pada perangkat elektronik semakin berukuran kecil dan menggunakan teknologi *multiprocessor*. Sehingga digunakan teknik *System on Chip* untuk mengatur hubungan antar prosesor - prosesor yang ada. *Network on Chip* ialah teknik yang digunakan di *System on Chip* sebagai pengganti *shared bus* dan *direct point – to – point*. Pada *Network on Chip* terdapat parameter desain dan parameter performansi jaringan. Penentuan parameter desain dan perkembangan dari jaringan dapat menimbulkan permasalahan pada jaringan seperti *congestion* dan saturasi yang menyebabkan paket hilang. *Congestion* dapat diatasi dengan menggunakan *flow control* yang tepat. Pada penelitian ini dilakukan analisis terhadap tiga teknik *flow control* yaitu Stall / Go , Ack / Nack serta Dynamic Multi Level yang diterapkan pada dua model jaringan. Model jaringan yang pertama untuk mengamati pengaruh *flow control* terhadap saturasi jaringan dan model kedua untuk mengamati pengaruh *flow control* terhadap perubahan parameter desain dan mendapatkan teknik *flow control* yang paling optimal dan pengaruh perubahan parameter desain terhadap parameter performansi jaringan. Dari hasil penelitian, jaringan yang menggunakan *flow control* tidak mengalami saturasi. Dimana *flow control* Stall / Go merupakan *flow control* terbaik dalam meningkatkan *throughput* sebesar 21.08% , 65.33%, 151% dan 13.37% , menurunkan *delay* sebesar 407.85, 606.03, 1631.95, 322.59 cycles, menurunkan penggunaan daya sebesar 68.67%, 61.33%, 49.93%, 68.22% untuk masing – masing perubahan ukuran jaringan, perubahan *packet injection rate*, perubahan ukuran paket dan perubahan ukuran *buffer*.**

Kata Kunci—Congestion , Flow Control , Network on Chip, System on Chip

I. PENDAHULUAN

SEMAKIN berkembangnya teknologi, ukuran dari perangkat keras yang digunakan pada perangkat elektronik berubah menuju ke ukuran yang semakin kecil. Dimana perangkat elektronik menggunakan teknologi multiprocessor dengan ukuran yang sangat kecil. Teknik *System on Chip* digunakan untuk mengatur hubungan antar processor – processor yang ada. Processor – processor yang digunakan pada SoC disebut sebagai *Processing Element* (PE). Dimana setiap PE dapat saling berkomunikasi satu sama lain. Teknik yang digunakan agar PE dapat saling berkomunikasi seperti *shared bus* dan *direct point – to – point*. Penggunaan kedua teknik tersebut memiliki kelebihan yaitu sederhana dan mudah dalam penerapannya.Tetapi seiiring dengan meningkatnya ukuran dari jaringan akan menimbulkan permasalahan seperti *bottleneck* pada *shared bus* dan meningkatnya kompleksitas dan *delay* pada *direct point – to – point*. Salah satu solusi yang dapat digunakan untuk mengatasi permasalahan tersebut adalah *Network on Chip* (NoC). Dimana NoC dapat meningkatkan reliabilitas , efisiensi dan *delay* interkoneksi pada jaringan.

NoC memiliki parameter desain dan parameter performansi jaringan. Dimana penentuan parameter desain yang tidak tepat dan perubahan dari jaringan dapat menimbulkan permasalahan pada jaringan seperti *congestion* dan saturasi. *Congestion* terjadi karena adanya overlaod pada penerima dimana adanya ketidak sesuaian antara ukuran buffer dan ukuran paket yang dikirimkan serta tidak diketahuinya informasi mengenai buffer penerima oleh pengirim. *Congestion* dapat menyebabkan paket yang diterima di drop yang berakibat pada *error* dan menurunnya performa jaringan. Jaringan yang tersaturasi akan menyebabkan throughput pada jaringan mencapai nilai maksimum meskipun parameter desain lainnya dirubah.

Congestion dan saturasi dapat diatasi dengan menerapkan *flow control* pada jaringan. *Flow control* digunakan untuk mencegah terjadinya antrian yang melebihi kapasitas, mengatur pengalokasian *buffer* dan titik pada setiap *router* dan menentukan kapan *buffer* dan link yang digunakan. Dalam memilih *flow control* harus diperhatikan pengaruh yang diberikan terhadap parameter desain seperti *overhead*, area yang digunakan dan pengkabelan dimana dapat mempengaruhi performansi jaringan.

Pada penelitian ini akan dilakukan analisis terhadap tiga buah teknik *flow control* yang ada pada NoC yaitu Stall / Go , Ack / Nack serta Dynamic Multi Level. Ketiga teknik *flow control* tersebut diterapkan pada dua model jaringan. Model jaringan yang pertama digunakan untuk mengamati pengaruh *flow control* terhadap saturasi jaringan. Sedangkan model jaringan yang kedua digunakan untuk mengamati pengaruh *flow control* terhadap perubahan parameter desain dan untuk mendapatkan teknik *flow control* yang paling optimal.

Adapun sistematika penulisan dalam paper ini adalah pada

bab 1 dijelaskan pendahuluan dan latar belakang. Bab 2 menjelaskan mengenai dasar teori yang berisi definisi – definisi yang digunakan dalam penelitian ini. Bab 3 akan dijelaskan mengenai mekanisme yang digunakan dalam mengatasi serangan pada jaringan. Bab 4 akan menampilkan hasil simulasi dan analisa performansi dari mekanisme yang digunakan. Bab 5 memberikan kesimpulan dari paper dan saran yang dapat dilakukan pada penelitian selanjutnya

II. DASAR TEORI

A. System on Chip

Pada SoC, beberapa sistem elektronik dan sistem komputasi yang disebut sebagai *Processing Element* (PE) ditanam pada sebuah chip. Teknologi yang digunakan pada komunikasi antar core yaitu *direct point – to – point , shared bus* dan *network on chip*.

Direct point – to – point merupakan teknik yang digunakan pertama kali pada SoC dimana setiap PE saling berkomunikasi satu sama lain melalui kabel yang menghubungkan setiap PE. *Direct point – to – point* memiliki skalabilitas yang rendah dan kompleksitas yang tinggi bila diterapkan pada jaringan yang besar. Namun untuk jaringan yang kecil , teknik ini memberikan performansi yang lebih baik.

Shared bus menggunakan *interface* untuk menghubungkan bus dengan setiap PE dan bus *arbiter* untuk mengatur proses komunikasi setiap PE. Teknik ini membutuhkan input dan output pin yang sedikit sehingga area untuk pengkabelan dan biaya yang diperlukan murah. Namun skalabilitas dari *shared bus* kurang baik dan proses pengiriman data lambat karena adanya *arbitration* dan jika terjadi *contention*.

Network on Chip memilliki infrastruktur yang fleksibel dan mengadopsi arsitektur dari jaringan komputer untuk mengirimkan dan menerima paket antar PE. NoC menggunakan router untuk meneruskan paket dari pengirim dan penerima. NoC menggunakan OSI model seperti jaringan komputer pada umumnya dimana tiga layer yang paling utama adalah layer fisik untuk sinyal clock dan sinyal kontrol, layer *data link* untuk *flitization* , *deflitization* ,*error detection* dan *correction* dan layer network untuk *packetization*, *routing*, *buffering*, *congestion detection* and *control*.

B. Arsitektur Network on Chip

Arsitektur dari NoC direpresentasikan sebagai kombinasi antara *Processing Element* (PE) , *Resource Network Interface* (RNI) dan juga router.

PE dapat berupa komponen hardware seperti processor, FPGA, Amlifier , DSP. Sedangkan RNI bertugas untuk menghubungkan PE dengan router sehingga setiap PE dapat mengirimkan pesan ke router tujuan. Router yang digunakan memiliki tugas utama untuk mentransmisikan pesan yang diterima PE ke tujuan apabila router terhubung secara langsung dengan PE tujuan atau meneruskan paket ke router tujuan. Router yang digunakan terdiri dari lima port yaitu Utara, Selatan, Barat , Timur yang terhubung dengan router lainnya dan port lokal yang terhubung dengan PE. Setiap router memiliki buffer yang berfungsi sebagai tempat penyimpanan sementara.

C. Parameter Desain dan Performansi Network on Chip

Dalam mendesain sebuah jaringan pada NoC , parameter desain yang harus diperhatikan yaitu topologi, teknik switching , dan algoritma routing dengan permasalahan utama yaitu bagaimana dapat menghubungkan setiap PE hingga dapat saling berkomunikasi satu sama lain dengan performansi yang maksimal.

Topologi dapat didefinisikan sebagai layout fisik yang mendeskripsikan bagaimana router pada NoC dapat terhubung satu sama lain dengan syarat skalabilitas tinggi dan konsumsi daya rendah. Dua jenis topologi yang sering digunakan adalah mesh dan torus. Teknik *switching* menentukan kapan paket yang datang harus dilayani oleh router dan menentukan kapan paket tersebut dikirimkan oleh router ke router berikutnya. Terdapat tiga jenis teknik *switching* yang sering digunakan yaitu *store and forward*, *wormhole switching* dan *virtual cut through*. Sedangkan algoritma routing menentukan rute yang akan dilalui oleh paket yang dikirimkan untuk mencapai tujuan. Algoritma routing dapat memberikan pengaruh terhadap kompleksitas desain router dan luas penggunaan area serta mempengaruhi konsumsi daya.

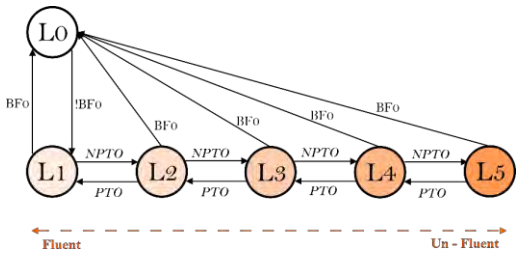
Parameter performansi pada NoC dapat berupa throughput, *delay* dan konsumsi daya pada jaringan. Dimana parameter performansi dipengaruhi oleh parameter desain yang digunakan.

Delay didefiniskan sebagai waktu antara saat PE sumber mengirimkan flit pertama ke jaringan dan saat PE tujuan menerima flit terakhir dari paket yang dikirimkan. *Delay* dihitung dalam satuan cycles. Sedangkan throughput dapat dikatakan sebagai laju pengiriman paket pada jaringan yang diterima oleh node tujuan dalam satuan waktu dan dinyatakan dengan satuan flits/cycle Sedangkan penggunaan daya pada NoC dihitung dalam satuan Joule yang merupakan daya total yang digunakan oleh semua komponen dan proses yang terjadi pada jaringan.

D. Flow Control

Flow control mengatur pengalokasian *buffer* dan *link* pada tiap router dan menentukan kapan *buffer* dan *link* digunakan untuk suatu paket. Pemilihan *flow control* harus sesuai dengan kebutuhan karena dapat mempengaruhi kompleksitas pengkabelan, luas area, *overhead* yang mempengaruhi performansi. Beberapa *flow control* yang dapat digunakan pada NoC yaitu *Flow Control* Stall / Go , Ack / Nack dan Dynamic Multi Level (DyML).

Flow Control Stall / Go memerlukan dua kabel untuk kontrol yaitu untuk *flagging data availability* yang berarah maju dan signalling kondisi *buffer* (Stall / Go) yang berarah mundur. Pada jaringan yang menerapkan *flow control* ini , setiap node mengetahui availabilitas dari slot kosong pada *buffer node downstream*. Saat slot kosong *node* tujuan mencapai batas off maka sinyal Stall akan dikirim ke node



Gambar II.1 Perindahan Level pada Dynamic Multi Level Flow Control

Tabel II.1 Tingkatan Buffer Fluidity dan Buffer Fill pada Dynamic Multi Level Flow Control

Level	***Buffer* Fluidity**	***Buffer* Fill**
L0	Empty (*buffer* fill = 0)	Buffer Fill = Buffer Size
L1	Fluent	Buffer Fill = Buffer Size - 1
L2	Middle Fluent	Buffer Fill > Buffer Size / 2
L3	Less Fluent	Buffer Fill > Buffer Size / 4
L4	Near Un – Fluent	Buffer Fill > Buffer Size / 8
L5	Un – Fluent	

upstream hingga mencapai node sumber dan sumber akan berhenti mengirimkan flit. Router pada node perantara saat meneriam sinyal Stall akan menyimpan flit yang datang di *output register* dan *flow control register* kemudian mengirimkan Sinyal Stall ke *node upstream* berikutnya. Node tujuan memproses data yang ada di *buffer* dan saat slot kosong mencapai batas On maka node tujuan akan mengirimkan sinyal Go ke node *upstream* hingga mencapai node sumber. Node sumber yanag menerima sinyal Go akan melanjutkan proses transmisi.

Pada *flow control* Ack / Nack , *node* sumber tidak mengetahui kondisi *buffer* pada node *downstream*, dimana node sumber akan mentransmisikan flit secara terus menerus ke node tujuan. Apabila node tujuan menerima flit maka akan mengirimkan sinyal Ack ke node sumber dan mengirimkan sinyal Nack apabila tidak menerima flit yang diharapkan. Apabila node sumber menerima Nack maka akan dilakukan pengiriman ulang.

Pada *flow control* DyML menggunakan konsep dasar dari *buffer fluidity* dimana terdapat enam kondisi yang digunakan dalam *flow monitor* untuk merepresentasikan kondsi jaringan sesuai dengan Gambar II.1. Dimana L0 hingga L5 merupakan tingkatan dari *buffer fluidity*, BF0 adalah kejadian saat *buffer fill* kosong sedangkan BF!0 saat *buffer* tidak kosong. NPTO adalah *Non Pop Time Out* dan PTO adalah *Pop Time Out* dimana *Pop* merupakan kejadian saat *buffer* meneruskan flit ke node berikutnya. Pada DyML, setiap tingkatan dipengaruhi oleh *buffer fluidity* dan *buffer fill* sesuai Tabel II.1

III. RANCANGAN JARINGAN DAN IMPLEMENTASI FLOW CONTROL

A. Model Jaringan Pertama

Pada model jaringan yang pertama digunakan untuk mengamati pengaruh *flow control* terhadap saturasi pada jaringan dengan parameter yang diamati titik saturasi dan *throughput* maksimum. Dimana saturasi pada jaringan terjadi dengan menggunakan kombinasi parameter – parameter desain yang sesuai. Sehingga digunakan dua skenario pengujian yaitu yang pertama kombinasi antara ukuran *buffer* dengan *packet injection rate* yang berubah dan skenario kedua kombinasi *packet injection rate* dan ukuran paket yang berubah. Dimana parameter desain yang nilainya tetap yaitu waktu simulasi 100000 cycles, topologi 2D Mesh , ukuran jaringan 5 x 5 , algoritma routing XY, *selection function random* , dan model trafik transpose. Sedangkan parameter desain yang berubah – ubah nilainya yaitu Ukuran buffer 2 – 10 flits, ukuran paket 2 – 22 flits , *packet injection rate* 0.02 – 0.3.

B. Model Jaringan Kedua

Model jaringan kedua digunakan untuk mengamati pengaruh perubahan parameter desain dan ketiga *flow control* terhadap parameter performansi. Dimana parameter performansi yang diamati adalah throughput, *delay* dan konsumsi daya. Pada model jaringan kedua terdapat 4 skenario yang digunakan. Skenario 1 adalah ukuran jaringan berubah – ubah nilainya dari 2 x 2 hingga 9 x 9. Skenario 2 adalah *packet injection rate* berubah – ubah dari 0.02 hingga 0.18. Skenario 3 adalah ukuran paket berubah – ubah dari 2 hingga 18 flit. Skenario 4 adalah ukuran *buffer* berubah – ubah dari 1 hingga 17 flit.

Adapun parameter desain untuk keseluruhan memiliki nilai tetap seperti waktu simulasi 100000 cycle , topologi 2D – Mesh, algoritma routing XY, *Selection function Random*, Model Trafik *Transpose*, *packet injection rate* 0.06, ukuran paket 2 – 4 flit, ukuran *buffer* 8 flit, ukuran jaringan 5 x 5.

C. Implementasi Flow Control

Setelah didapat algoritma untuk masing – masing metode *flow control* dan dengan model jaringan yang telah ada dapat dilakukan proses pengimplementasian algoritma kedalam Noxim sebagai simulator yang digunakan pada penelitian ini. Dikarenakan Noxim merupakan simulator yang menggunakan SystemC, maka algoritma yang sudah ada harus diubah ke dalam bentuk bahasa C++ terlebih dahulu dan diterapkan pada file – file tertentu pada Noxim. Untuk dapat mengimplementasikan *flow control* pada Noxim diperlukan adanya perubahan pada file *source* Noxim. Sehingga file – file *source* Noxim yang harus dirubah untuk dapat mengimplementasikan ketiga teknik *flow control* tersebut yaitu:

- NoximProcessingElement untuk mengatur pengiriman dan penerimaan flit di *Processing Element* tiap *router*
- NoximRouter untuk mengatur proses pengiriman dan penerimaan flit dan proses *routing* pada *router*
- NoximTile dan NoximNoC untuk pengaturan kanal arah maju dan mundur yang digunakan pada NoC
- NoximMain untuk melakukan pendefinisian parameter *global* yang digunakan
- NoximBuffer untuk mengatur bagaimana operasi – operasi pada *buffer*

IV. HASIL DAN ANALISA

A. Analisis Pengaruh Flow Control terhadap Saturasi di Jaringan

Dilakukan analisis dengan parameter yang diamati merupakan titik saturasi dan throughput maksimum.

Untuk skenario pertama yang menggunakan kombinasi ukuran *buffer* dan *packet injection rate* yang berubah – ubah hasil yang diperoleh sesuai dengan Tabel IV.1. Untuk jaringan yang hanya menggunakan *flow control default*, jaringan mengalami saturasi dikarenakan pengirim hanya dapat mengirimkan flit dengan *sliding window* 1 dan memerlukan *signalling* dari penerima sebelum dapat mengirimkan flit berikutnya. Sehingga dengan ukuran *buffer* dan ukuran flit yang tetap, sedangkan *packet injection rate* terus diperbesar akan dicapai kondisi dimana jumlah flit yang semakin banyak di jaringan tidak akan meningkatkan *throughput*, Sedangkan pada jaringan yang menerapkan *flow control* Stall / Go , Ack / Nack dan DyML tidak menyebabkan saturasi pada jaringan dikarenakan jaringan dapat mengirimkan flit secara terus menerus dan tidak terbatasi oleh *sliding window*. Sehingga apabila jumlah paket yang dikirimkan semakin banyak sebagai akibat dari peningkatan throughput, tidak akan menyebabkan saturasi.

Untuk skenario kedua yang menggunakan kombinasi *packet injection rate* dan ukuran paket dengan mengacu pada hasil yang diperoleh sesuai Tabel IV.2 jaringan yang menggunakan *flow control* default mengalami saturasi. Sedangkan untuk *flow control* Stall / GO tidak terjadi saturasi dengan peningkatan ukuran paket hingga 20 – 22 flit. Dikarenakan flit dapat dikirimkan terus menerus selama *signalling* bertanda GO dan karena perubahan *signalling* menjadi STALL dipengaruhi oleh ukuran *buffer* yang mempengaruhi batas off dan batas on.

Tabel IV.1 Perbandingan Throughput Maksimum dan Titik Saturasi dengan kombinasi ukuran buffer dan packet injection rate yang berubah

No	*Flow Control*	*Throughput* Maksimum	Titik Saturasi (PIR)
1	Default	0.25742 flits / Cycle	0.18
2	Stall / Go	Tidak Tersaturasi	Tidak Tersaturasi
3	Ack / Nack	Tidak Tersaturasi	Tidak Tersaturasi
4	Dynamic Multi Level	Tidak Tersaturasi	Tidak Tersaturasi

Tabel IV.2 Perbandingan Throughput Maksimum dan Titik Saturasi dengan kombinasi packet injection rate dan ukuran flit yang berubah

No	*Flow Control*	*Throughput* Maksimum	Titik Saturasi (Ukuran Paket)
1	Default	0.25742 flits / Cycle	8 – 10 flit
2	Stall / Go	Tidak Tersaturasi	Tidak Tersaturasi
3	Ack / Nack	Tidak Tersaturasi	Tidak Tersaturasi
4	Dynamic Multi Level	Tidak Tersaturasi	Tidak Tersaturasi

Sedangkan pada flow control DyML dan Ack/Nack saat ukuran paket 8 – 10 flit terjadi throughput maksimum, namun saat ukuran paket diperbesar tidak terjadi saturasi melainkan throughput akan menurun. Hal tersebut disebabkan untuk jaringan dengan flow control Ack / Nack, flit dapat terus dikirimkan selama menerima Ack. Akan tetapi saat terjadi flit *drop* dan dikirimkan Nack harus dilakukan pengiriman ulang. Karena *buffer* yang digunakan hanya berukuran 8 flit saja maka saat ukuran paket yang digunakan lebih besar dari 8 – 10 flit kemungkinan untuk terjadinya flit *drop* dan pengiriman ulang semakin besar sehingga menyebabkan menurunnya throughput sesaat.

Sedangkan pada *flow control* DyML proses pengiriman flit bergantung pada kondisi *signalling* dan kondisi *buffer*. Dengan ukuran *buffer* yang tetap 8 flit kemungkinan *signalling* berada pada kondisi Stall untuk memproses flit yang terdapat di *buffer* akan semakin sering terjadi. Selain itu penentuan batas Off dan batas on serta rentang waktu pengeceakan *Pop* juga menjadi faktor yang menyebabkan menurunnya throughput.

B. Analisis Pengaruh Parameter Desain dan Flow Control terhadap Performansi Jaringan

Pada model jaringan kedua dilakukan analisis pengaruh parameter desain dan *flow control* terhadap performansi jaringan berupa *throughput* , *delay* dan konsumsi daya dengan empat skenario berbeda. Hasil dari penelitian yang dilakukan untuk model kedua dapat dilihat pada Tabel IV.3, Tabel IV.4, dan Tabel IV.5.

Untuk skenario pertama dimana dengan ukuran jaringan yang berubah terhadap *throughput*, untuk jaringan dengan *flow control* default, *flow control* Ack/Nack dan *flow control* DyML, ukuran jaringan berbanding terbalik dengan throughput.. Sedangkan untuk *flow control* Stall / Go, dengan meningkatkan ukuran jaringan tidak menyebabkan perubahan throughput yang signifikan. Sehinga *flow control* Stall / Go memberikan peningkatan throughput terbesar yaitu 21.08% jika dibandingkan dengan *flow control default*.

Untuk skenario kedua dengan *packet injection rate* yang berubah terhadap throughput, untuk semua jaringan yang menggunakan teknik *flow control* memberikan peningkatan throughput apabila *packet injection rate* ditingkatkan. Dengan melakukan perbandingan peningkatan throughput rata – rata masing – masing *flow control* terhadap *flow control default*, *flow control* Stall / Go memberikan peningkatan throughput terbaik sebesar 65.33%.

Untuk skenario ketiga dengan ukuran paket yang berubah terhadap throughput, untuk semua jaringan yang menggunakan teknik *flow control* memberikan peningkatan throughput apabila ukuran paket ditingkatkan. Dengan melakukan perbandingan peningkatan throughput rata – rata masing – masing *flow control* terhadap *flow control default*, *flow control* Stall / Go memberikan peningkatan throughput terbaik sebesar 151%.

Untuk skenario keempat dengan ukuran *buffer* yang berubah terhadap throughput, untuk jaringan yang menggunakan *flow control* Ack / Nack dan DyML ukuran *buffer* berbanding lurus

Tabel IV.3 Peningkatan Throughput (%) terhadap flow control default

	Parameter Desan			
Flow Control	Ukuran Jaringan	PIR	Ukuran Paket	Ukuran Buffer
Ack / Nack	14.96	59.21	53.19	2.54
Stall / Go	21.08	65.33	151	13.37
DyML	14.96	60.33	53.61	6.36

Tabel IV.4 Penurunan *Delay* (cycles) terhadap flow control default

	Parameter Desan			
Flow Control	Ukuran Jaringan	PIR	Ukuran Paket	Ukuran Buffer
Ack / Nack	403.52	601.63	1617.63	317.19
Stall / Go	407.85	606.03	1631.95	322.59
DyML	403.59	601.74	1611.35	316.86

Tabel IV.5 Penurunan Konsumsi Daya (%) terhadap flow control default

	Parameter Desan			
Flow Control	Ukuran Jaringan	PIR	Ukuran Paket	Ukuran Buffer
Ack / Nack	23.54	7.79	42.71	20.69
Stall / Go	68.67	61.33	49.93	68.22
DyML	25.41	9.32	43.27	19.77

dengan throughput. Sedangkan untuk jaringan yang menggunakan *flow control* Stall / Go dan Default, ukuran *buffer* tidak memberikan pengaruh yang signifikan terhadap perubahan *throughput*. Dimana dengan melakukan perbandingan terhadap *flow control default*, *flow control* Stall / Go memberikan peningkatan *throughput* terbaik sebesar 13.37%.

Untuk skenario pertama dimana dengan ukuran jaringan yang berubah terhadap *delay*, untuk jaringan dengan *flow control default*, ukuran jaringan berbanding lurus dengan *delay*. Sedangkan untuk *flow control* Stall / Go, ukuran jaringan tidak memberikan pengaruh yang signifikan terhadap *delay*. Pada *flow control* Ack/ Nack dan DyML , peningkatan ukuran jaringan berbanding lurus dengan *delay* hanya sampai ukuran jaringan 5x5. Sehinga *flow control* Stall / Go memberikan penurunan *delay* terbesar yaitu 407.85 cycles jika dibandingkan dengan *flow control default*.

Untuk skenario kedua dengan *packet injection rate* yang berubah terhadap *delay*, jaringan yang menggunakan *flow control default* dan Stall/Go, *packet injection rate* berbanding lurus dengan *delay*. Sedangkan untuk *flow control* Ack/Nack dan DyML, *delay* yang terjadi mengalami penurunan hingga titik tertentu kemudian *delay* akan naik. Dengan melakukan perbandingan peningkatan throughput rata – rata masing – masing *flow control* terhadap *flow control default*, *flow control* Stall / Go memberikan penurunan *delay* terbaik sebesar 606.03 cycles.

Untuk skenario ketiga dengan ukuran paket yang berubah terhadap *delay*, untuk semua jaringan yang menggunakan teknik *flow control* memberikan penurunan *delay* apabila ukuran paket ditingkatkan. Dengan melakukan perbandingan penurunan *delay* rata – rata masing – masing *flow control* terhadap *flow control default*, *flow control* Stall / Go memberikan peningkatan throughput terbaik sebesar 1631.95 cycles.

Untuk skenario keempat dengan ukuran *buffer* yang berubah terhadap *delay* untuk jaringan yang menerapkan *flow control default* dan *flow control* Stall perubahan ukuran *buffer* tidak memberikan pengaruh signifikan terhadap perubahan *delay*.Sedangkan untuk *flow control* Ack/Nack dan DyML ukuran *buffer* berbanding lurus dengan *delay*. Dimana dengan melakukan perbandingan terhadap *flow control default*, *flow control* Stall / Go memberikan penurunan *delay* terbaik sebesar 322.59 cycles

Untuk skenario pertama dimana dengan ukuran jaringan yang berubah terhadap konsumsi daya, untuk semua jaringan dengan *flow control* ukuran jaringan berbanding lurus dengan konsumsi daya. Sehinga *flow control* Stall / Go memberikan penurunan konsumsi daya terbesar yaitu 68.67% jika dibandingkan dengan *flow control default*.

Untuk skenario pertama dimana dengan ukuran jaringan yang berubah terhadap konsumsi daya, untuk semua jaringan dengan *flow control* ukuran jaringan berbanding lurus dengan konsumsi daya. Sehinga *flow control* Stall / Go memberikan penurunan konsumsi daya terbesar yaitu 61.33% jika dibandingkan dengan *flow control default*.

Untuk skenario ketiga dengan ukuran paket yang berubah terhadap konsumsi daya, untuk jaringan yang menggunakan *flow control default*, saat ukuran paket yang digunakan adalah 8 – 10 flit mulai terjadi saturasi pada konsumsi daya. Sedangkan untuk jaringan yang menggunakan *flow control* Stall / Go , Ack /Nack dan DyML konsumsi daya berbanding lurus dengan ukuran paket. Dimana dengan melakukan perbandingan terhadap *flow control default*, *flow control* Stall / Go memberikan penurunan konsumsi daya terbaik sebesar 49.93%.

Untuk skenario keempat dengan ukuran buffer yang berubah terhadap konsumsi daya, untuk jaringan yang menggunakan *flow control* Stall / Go dan *default* , peningkatan ukuran buffer tidak berdampak pada konsumsi daya. Sedangkan untuk *flow control* Ack/Nack dan DyML ukuran *buffer* berbanding lurus dengan konsumsi daya. Dimana dengan melakukan perbandingan terhadap *flow control default*, *flow control* Stall / Go memberikan penurunan konsumsi daya terbaik sebesar 68.22%.

V. KESIMPULAN/RINGKASAN

Kesimpulan yang dapat diambil dari penelitian ini yaitu Pada model jaringan pertama , *flow control* Stall / Go , Ack / Nack dan DyML tidak menyebabkan jaringan mengalami saturasi hingga *packet injection rate* yang digunakan sebesar 0.3 untuk kombinasi pertama dan hingga ukuran paket yang digunakan sebesar 20 – 22 flit untuk kombinasi kedua.

Pada model jaringan kedua, throughput dapat ditingkatkan dengan memperbesar *packet injection rate*, ukuran paket, ukuran *buffer* dan ukuran jaringan. *Delay* dapat diturunkan

dengan memperkecil *packet injection rate*, ukuran paket, ukuran jaringan dan memperbesar ukuran *buffer*. Penggunaan daya pada jaringan dapat dikurangi dengan memperkecil *packet injection rate*, ukuran paket , ukuran *buffer* dan ukuran jaringan.

Pada model jaringan kedua, teknik *flow control* Stall / Go memberikan hasil terbaik dalam peningkatan throughput rata – rata jika dibandingkan dengan *flow control default* sebesar 21.08% terhadap perubahan ukuran jaringan , 65.33% terhadap perubahan *packet injection rate*, 151% terhadap perubahan ukuran paket dan 13.37% terhadap perubahan ukuran *buffer*. teknik *flow control* Stall / Go memberikan hasil terbaik dalam penurunan *delay* rata – rata jika dibandingkan dengan *flow control default* sebesar 407.85 cycles terhadap perubahan ukuran jaringan , 606.03 cycles terhadap perubahan *packet injection rate*, 1631.95 cycles terhadap perubahan ukuran paket dan 322.59 cycles terhadap perubahan ukuran buffer. Selain itu teknik *flow control* Stall / Go memberikan hasil terbaik dalam penurunan penggunana daya rata – rata jika dibandingkan dengan *flow control default* sebesar 68.67 % terhadap perubahan ukuran jaringan , 61.33% terhadap perubahan *packet injection rate*, 49.93% terhadap perubahan ukuran paket dan 68.22% terhadap perubahan ukuran *buffer*.

Saran untuk penelitian kedepannya dapat dilakukan perbandingan untuk kombinasi parameter desain yang lain untuk dapat mengetahui pengaruhnya terhadap kondisi saturasi pada jaringan. Dapat pula dilakukan implementasi model jaringan yang menggunakan flow control dan tidak menggunakan flow control pada perangkat hardware.

UCAPAN TERIMA KASIH

Penulis Muhammad Hizrian H mengucapkan terima kasih kepada Bapak Istas Pratomo dan Bapak Djoko Suprajitno atas Ilmu dan Bantuan yang telah diberikan untuk mengerjakan penelitian ini.

DAFTAR PUSTAKA

[1] J. Howard, "A 48 – Core IA – 32 Processor in 45 – nm CMOS Using on – Die Message – Passing and DVFS for Performance and Power Scaling", IEEE Journal of Solid State Circuits, Vol. 46 no1, pp – 173 – 183, jan 2011

[2] J. Kumar, A. Kanta, T. N. Kamal, K. K. Mahaptra, "Performance Evaluation of Different Routing Algorithm in Network on Chip", IEEE Asia Pacific Conference on Post Graduate Research in Microelectronics and Electronics, 2013

[3] R. Marculescu, U. Y. Orgras, L. S. Peh, N. E. Jerger, Y. Hosskote, "Outstanding Research Problems in NoC Design : System Microarchitecture", IEEE Trans. Computer – aided Design of Integrated Circuits System, Vol. 28, No 1, pp 3 – 21 Jan, 2009

[4] W. C. Tsai, Y. C. Lan, S. J. Chen, Y. Hen Hu, "DyML : Dynamic Multi Level Flow Control for Network on Chip", IEEE, 2010

[5] Williams James Dally and Brian Towles, "Principles and Practices of Interconnection Networks", Morgan Kaufmaan, 2004

[6] V. V. Nimbalkar, K. Varghese, "in – Channel Flow Control Scheme for Network on Chip", IEEE 13th Euromicro Conference on Digital System Design Architecture, Methods and Tools, 2010

[7] J. Hu, R. Marculescu, "DyAD – Smart Routing for Networks on Chip" Proceedings 41st Design Automation Conference, 2004

[8] W. Zhang, L. Hou, J. Wang, S. Geng, W. Wu, "Comparison Research between XY and Odd – Even Routing Algorithm of a 2 – Dimension 3 x 3 Mesh Topology Network on Chip" , IEEE Global Congress on Inteligent Systems, 2009

[9] XIAO Canwen, Z. Minxuan, D. Yong, Z. Zhitong, "Dimensional Bubble Flow Control and Fully Adaptive Routing in the 2 – D Mesh Network on Chip", IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008

[10] A. Pullini, F. Angielini, D. Bertozzi, L. Benini, "Fault Tolerance Overhead in Networks on Chip Flow Control Schemes", ACM, SBCCI, Brazil, September 4- 7

[11] Y.C. Lan, M.C. Chen, A.P. Su, Y. H. Hu, S. J. Chen, "Fluidity Concept for NoC : A *Congestion* Avoidance and Relief Routing Scheme", IEEE 21st International SoC Conference, pp 65 – 70, 2008

[12] M. Bakhouya, A. Chariete, J.Gaber, M. Wack, S. Niar, E. Coatanea, "Performance Evaluation of a Flow – Control Algorithm for Network on Chip" , IEEE, 2012

[13] I.Pratomo, "Adaptive NoC for Reconfigurable SoC"

[14] C. Nicopoulos, V. Narayanan, C. R. Das, "Network on Chip Architectures : A Holistic Design Exploration", Springer, 2009

[15] F. Gebali, H. Elmiligi, M. W. El – Kharashi, "Network on Chips : Theory and Practice", CRC Press, 2009

SIDANG TUGAS AKHIR

TELKOM #112

ANALISIS UNJUK KERJA FLOW CONTROL PADA NETWORK ON CHIP DALAM BERBAGAI KONDISI JARINGAN

M HIZRIAN H

DR. ISTAS
PRATOMO ST. MT

IR DJOKO
SUPRAJITNO R, MT

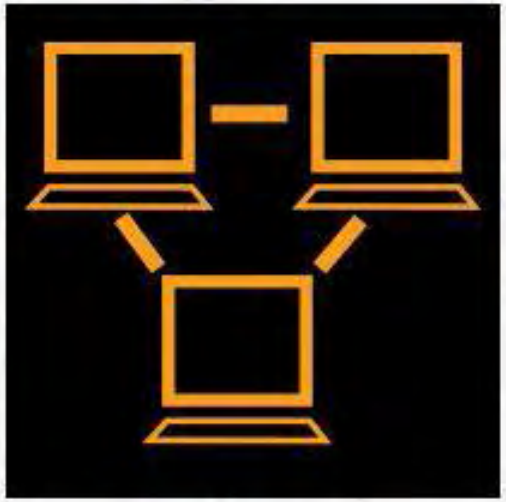
Network on Chip Flow Control

EARLY 2000

Vs

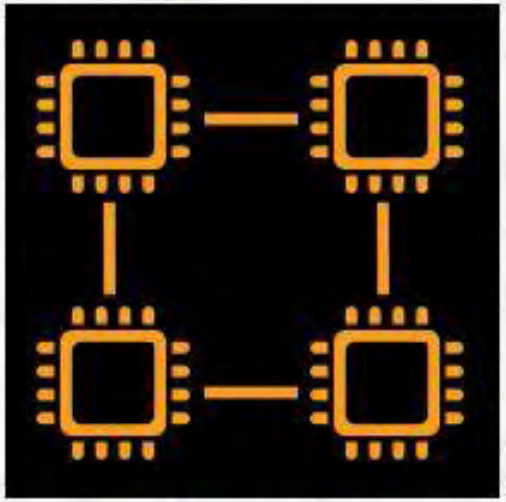
TODAY

NETWORK ON CHIP



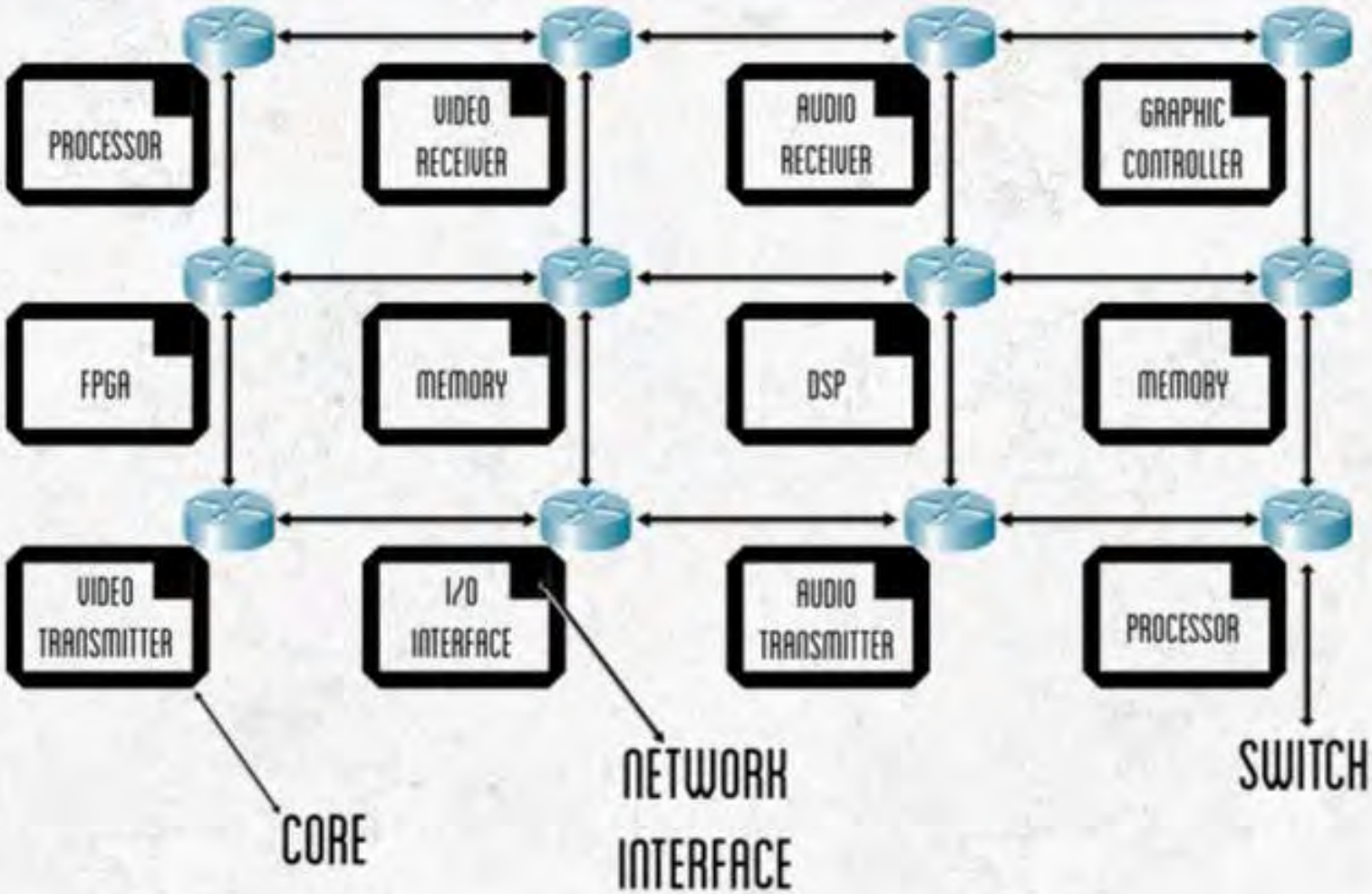
Jaringan Komputer

=



System on Chip

NETWORK ON CHIP



NETWORK ON CHIP

Parameter DESAIN

- Topologi Jaringan
- Packet Injection Rate
- Ukuran Paket
- Algoritma Routing
- Ukuran Buffer
- Flow Control

NETWORK ON CHIP

Performa IDAMAN

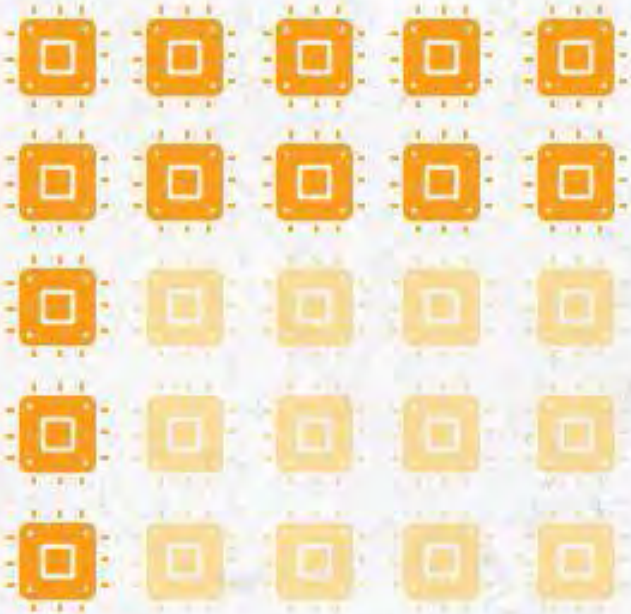
Throughput

Reliability

Delay

Konsumsi Daya

Luas Area



NETWORK ON CHIP

Karena

Parameter Desain yang tidak Sesuai

Menyebabkan

Congestion Saturasi

NETWORK ON CHIP

Sehingga

Performansi Menurun

Solusi

Flow Control

NETWORK ON CHIP

Flow Control

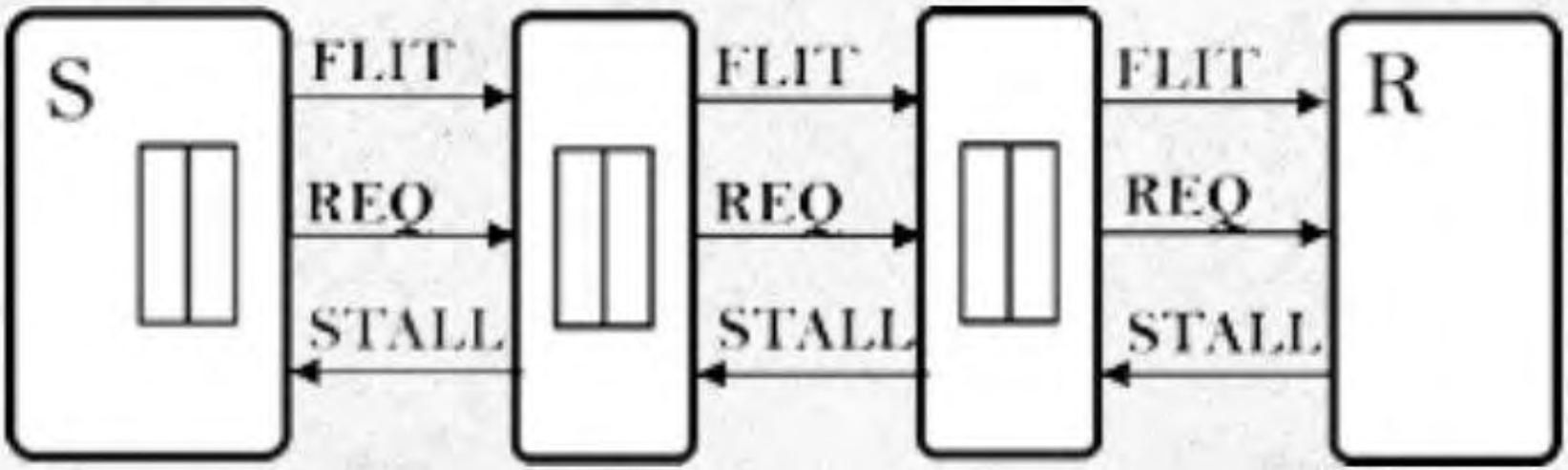
STALL / GO

ACK / NACK

Dynamic Multi Level

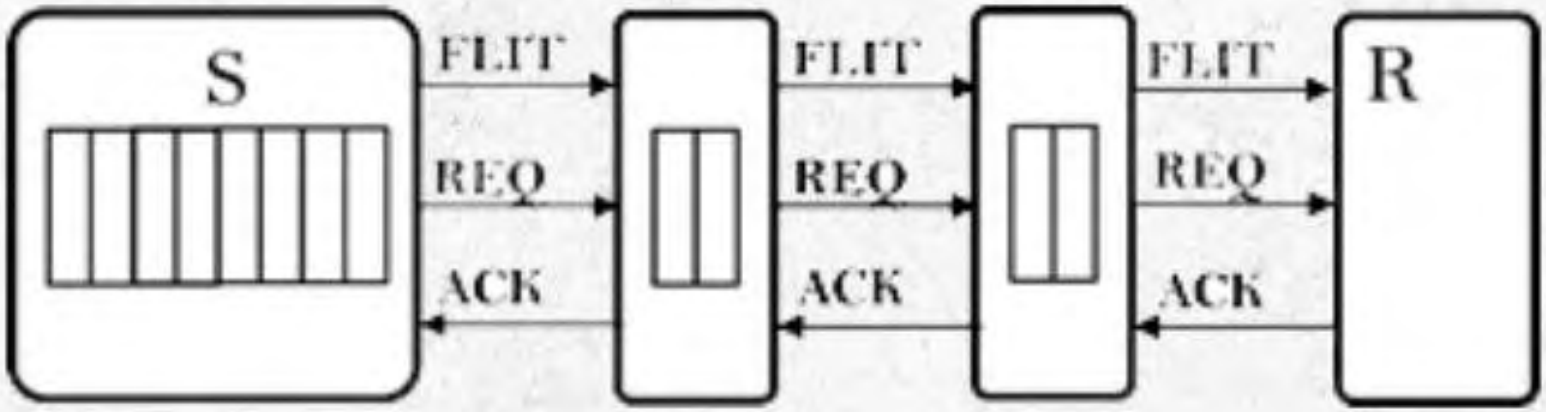
FLOW CONTROL

- STALL / GO



FLOW CONTROL

- ACK / NACK



FLOW CONTROL

- Dynamic Multi Level

"Pengiriman STALL / GO Berdasarkan pada Buffer Fill Level dan Buffer Fluidity Level"

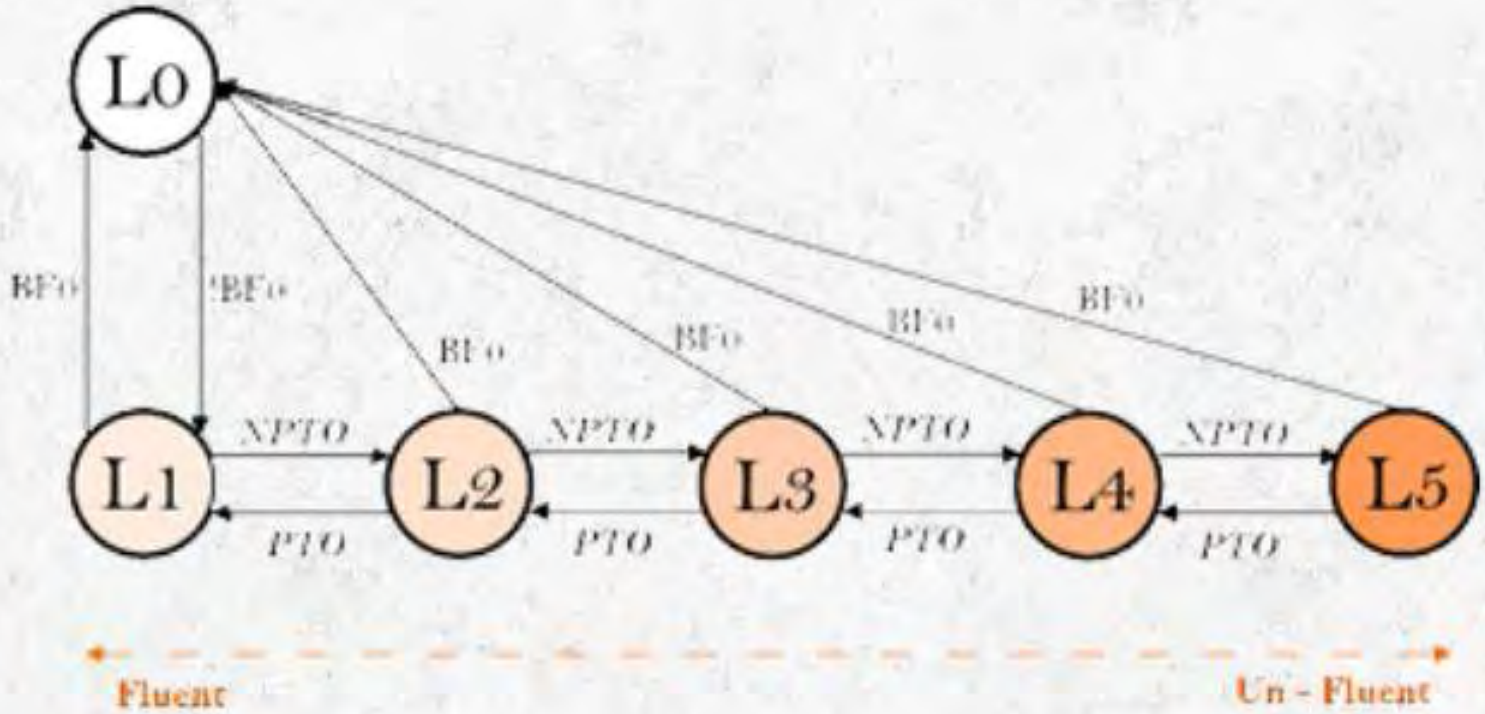
FLOW CONTROL

Dynamic Multi Level

Level	*Buffer* Fluidity	*Buffer* Fill
L0	Empty (*buffer* fill = 0)	Buffer Fill = Buffer Size
L1	Fluent	Buffer Fill = Buffer Size - 1
L2	Middle Fluent	Buffer Fill > Buffer Size / 2
L3	Less Fluent	Buffer Fill > Buffer Size / 4
L4	Near Un – Fluent	Buffer Fill > Buffer Size / 8
L5	Un – Fluent	

FLOW CONTROL

- Dynamic Multi Level



FLOW CONTROL

- Dynamic Multi Level

STALL :

Buffer Fill L0
Buffer Fill L1 & Buffer Fluidity L1
Buffer Fill L2 & Buffer Fluidity L2
Buffer Fill L3 & Buffer Fluidity L3
Buffer Fill L4 & Buffer Fluidity L4
Buffer Fluidity L5

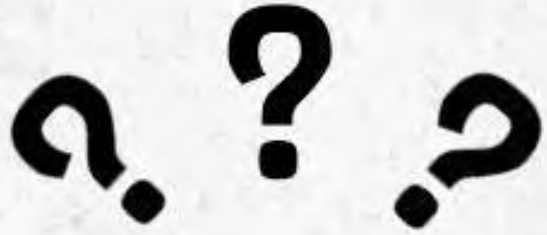
RUMUSAN MASALAH

Pengaruh Flow Control terhadap Saturasi pada jaringan

? ? ?

RUMUSAN MASALAH

Pengaruh Flow Control terhadap performansi jaringan



RUMUSAN MASALAH

Flow Control terbaik dalam meningkatkan performansi jaringan

METODOLOGI


Pembuatan Model jaringan
Penulisan Program Flow Control
Implementasi Program ke Simulator
Pengambilan Data
Analisa Hasil
Penarikan Kesimpulan

MODEL

Jaringan

MODEL JARINGAN

MODEL
Jaringan

- MODEL 1
  (Saturasi Jaringan)
- MODEL 2
  (Performa Jaringan)

MODEL JARINGAN 1

SKENARIO

- Ukuran Buffer dan Packet Injection Rate berubah
- Packet Injection Rate dan Ukuran Paket berubah

MODEL JARINGAN 1

PARAMETER DIUJI

- Titik Saturasi
- Throughput Maksimum

MODEL JARINGAN 2

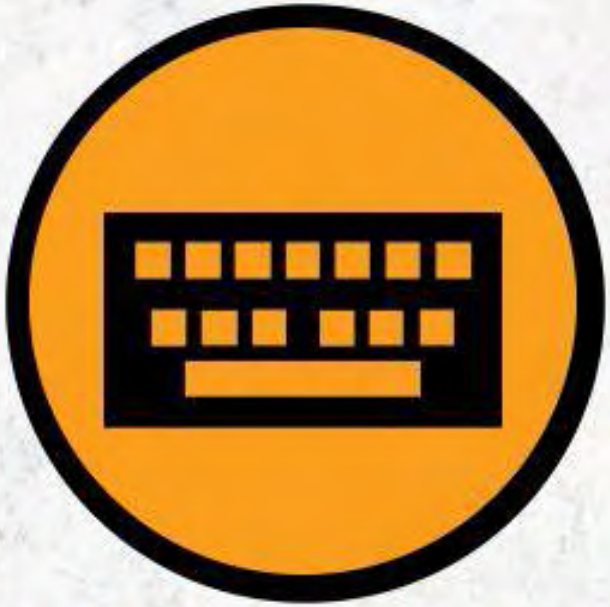
SKENARIO

- Ukuran Jaringan berubah
- Packet Injection Rate berubah
- Ukuran Paket berubah
- Ukuran Buffer berubah

MODEL JARINGAN

Parameter Diuji

- Throughput (flits/cyles)
- Delay (cycles)
- Konsumsi Daya (Joules)



SIMULASI PROGRAM

SIMULASI PROGRAM

NOXIM SIMULATOR

- SystemC
- Merubah File Source untuk Setiap Program Flow Control

SIMULASI PROGRAM

STALL/GO

- Batas Off = 80% dari Ukuran Buffer
- Batas On = 20% dari Ukuran Buffer

SIMULASI PROGRAM

DyML

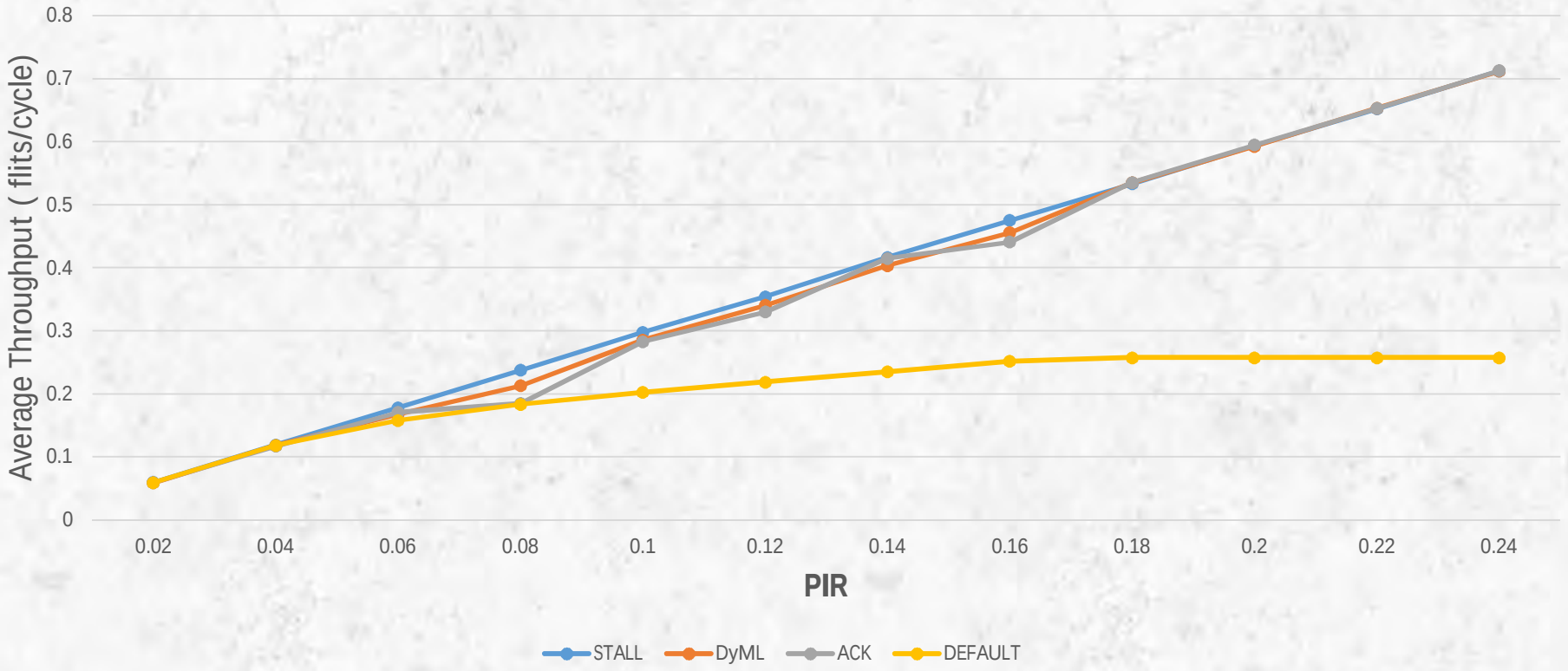
- Batas Off = 90% dari Ukuran Buffer
- Batas On = 50% dari Ukuran Buffer
- Jumlah Pop = 20% dari Ukuran Buffer
- Interval Check = 1% dari total waktu simulasi

HASIL
ANALISA

HASIL Model Jaringan 1

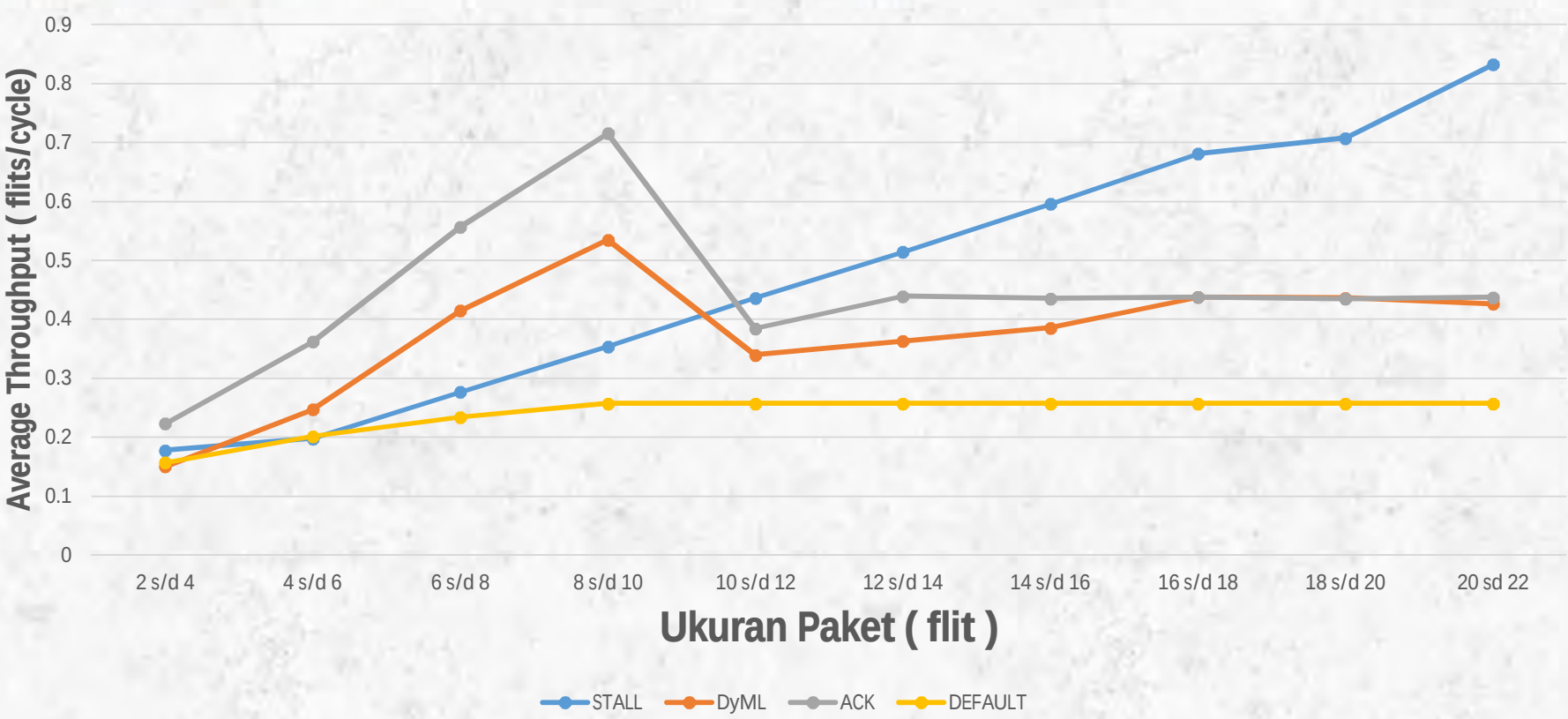
Skenario1

Pengaruh PIR dan Ukuran Buffer terhadap Throughput



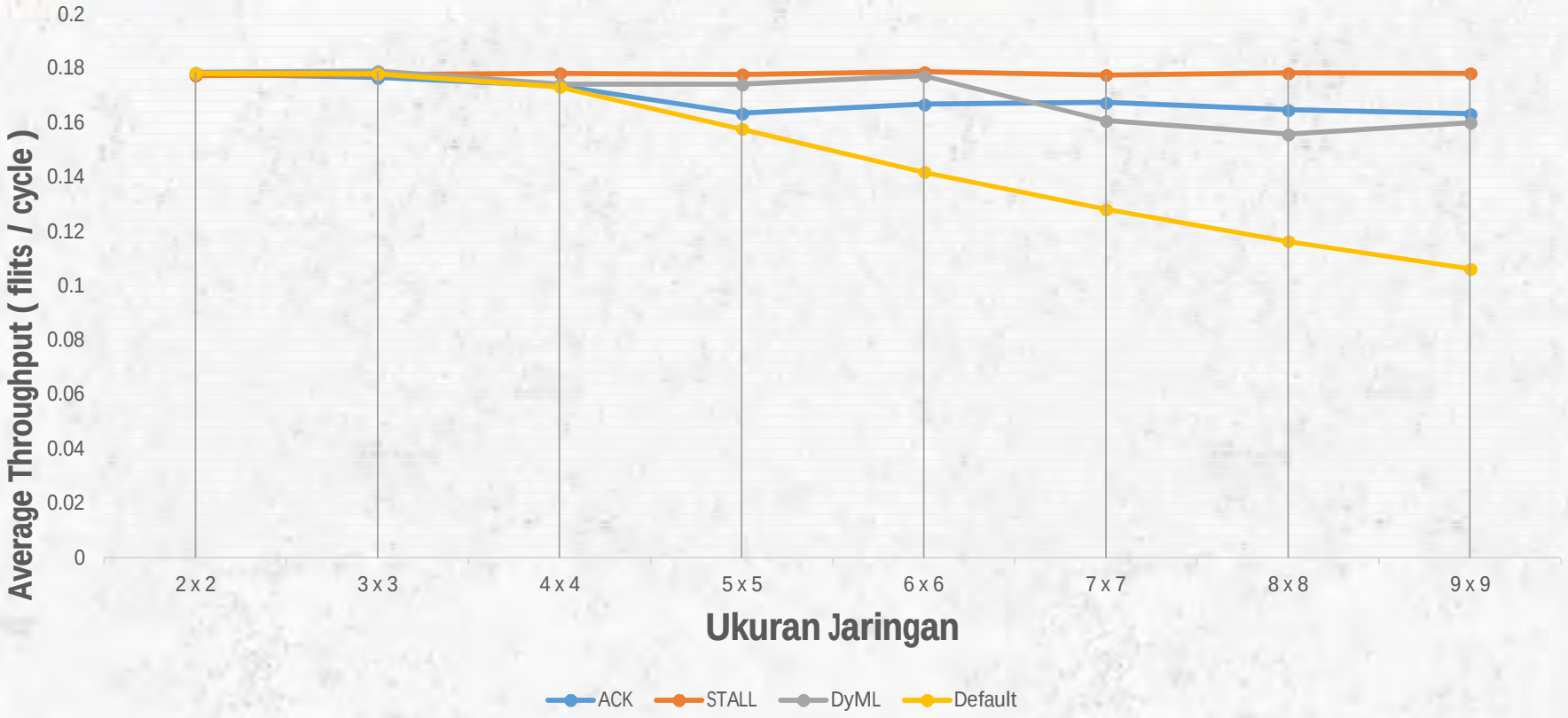
HASIL Model Jaringan 1

Skenario2



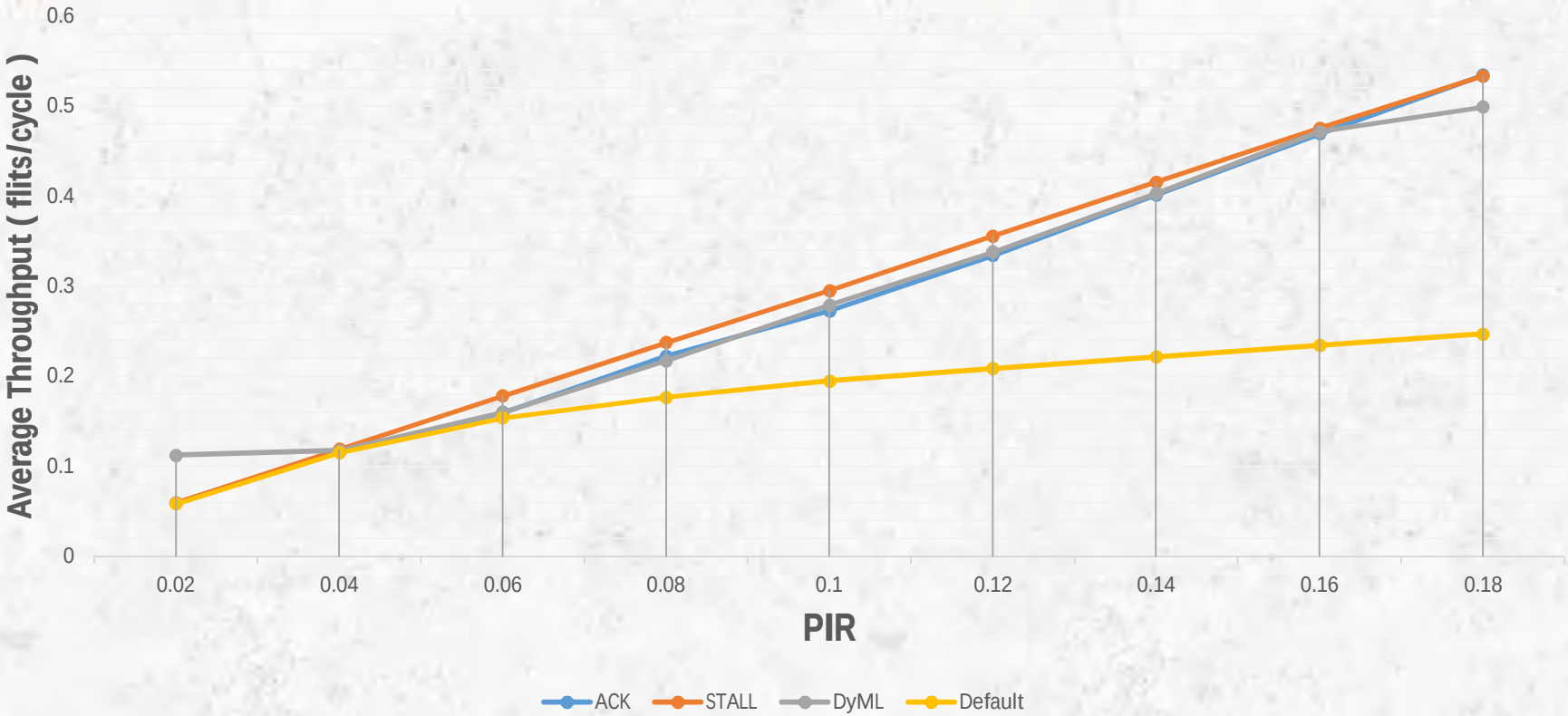
HASIL Model Jaringan 2

Perubahan THROUGHPUT



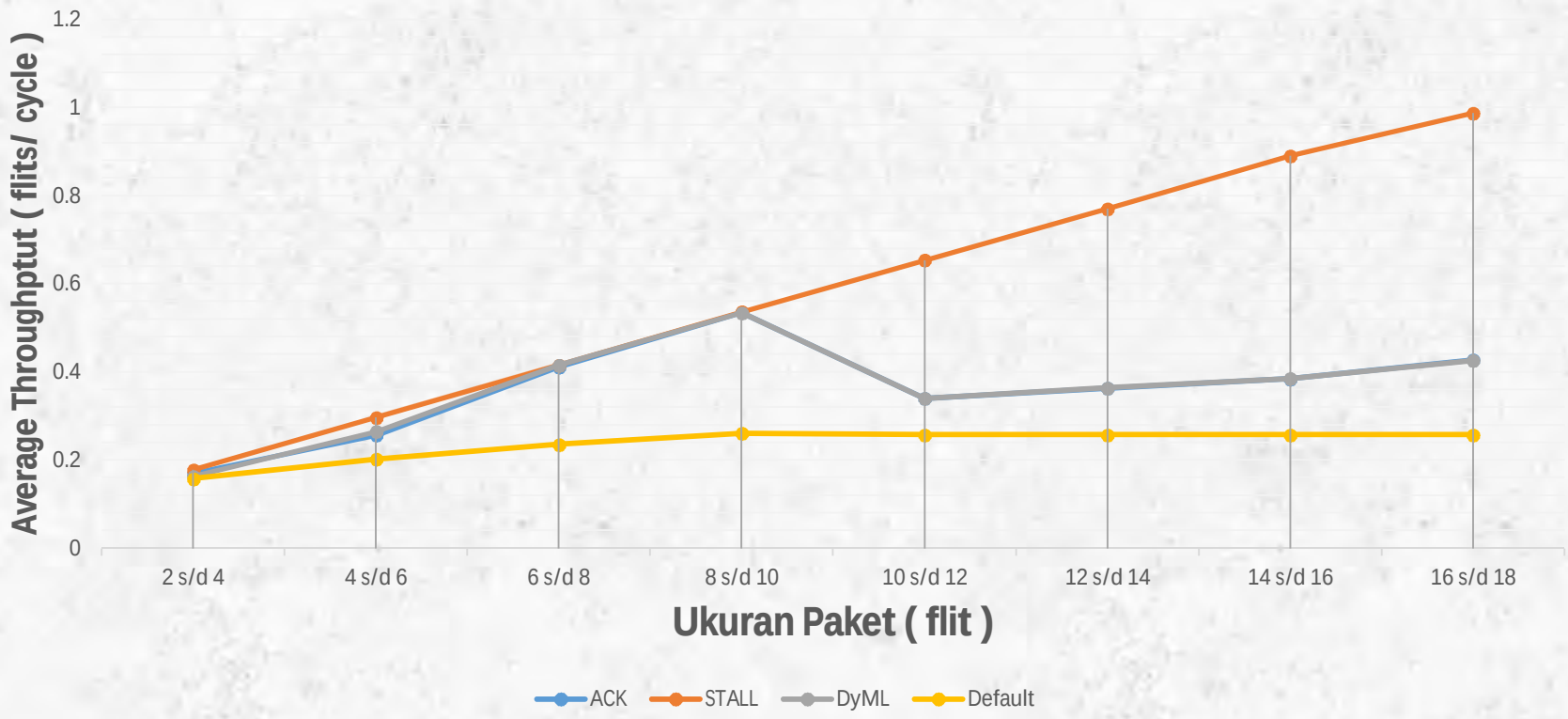
HASIL Model Jaringan 2

Perubahan THROUGHPUT



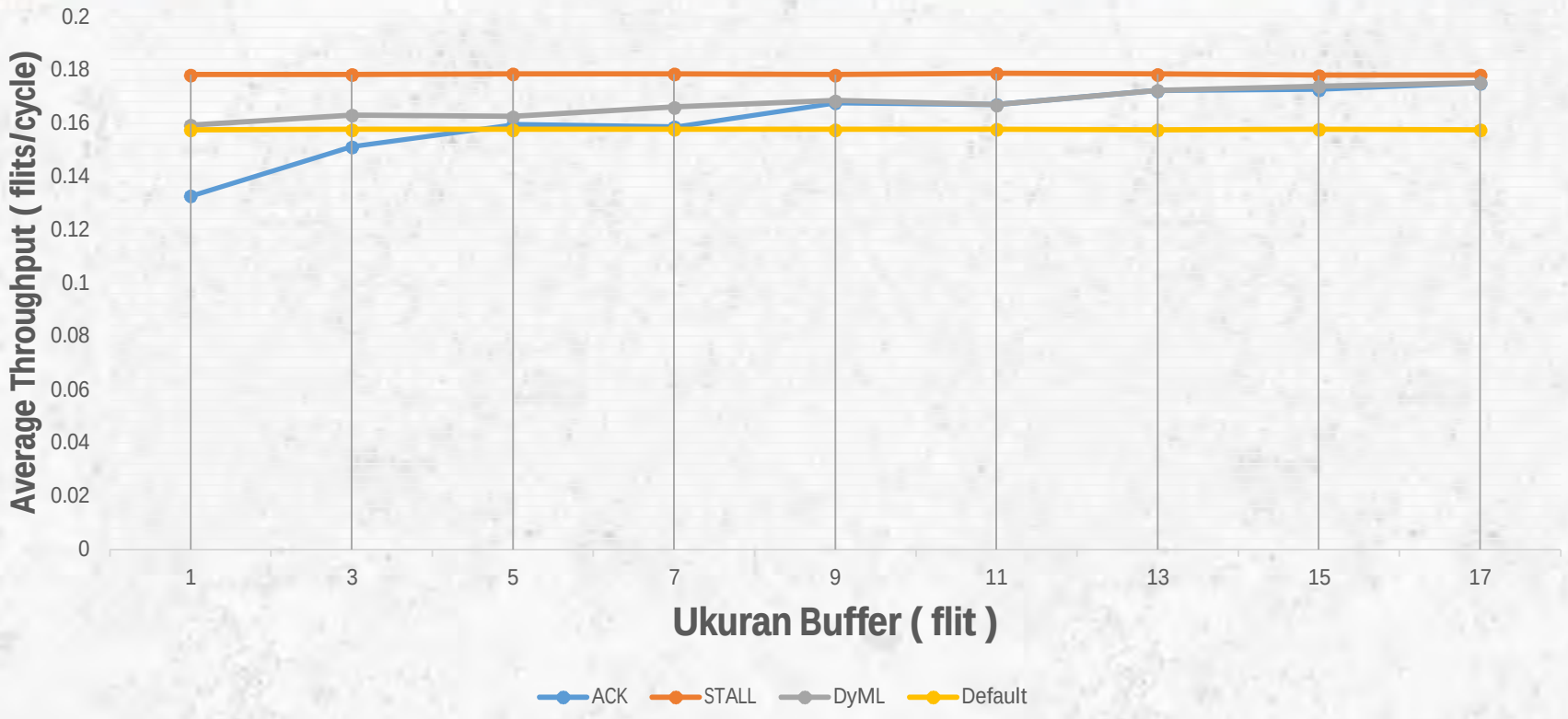
HASIL Model Jaringan 2

Perubahan THROUGHPUT



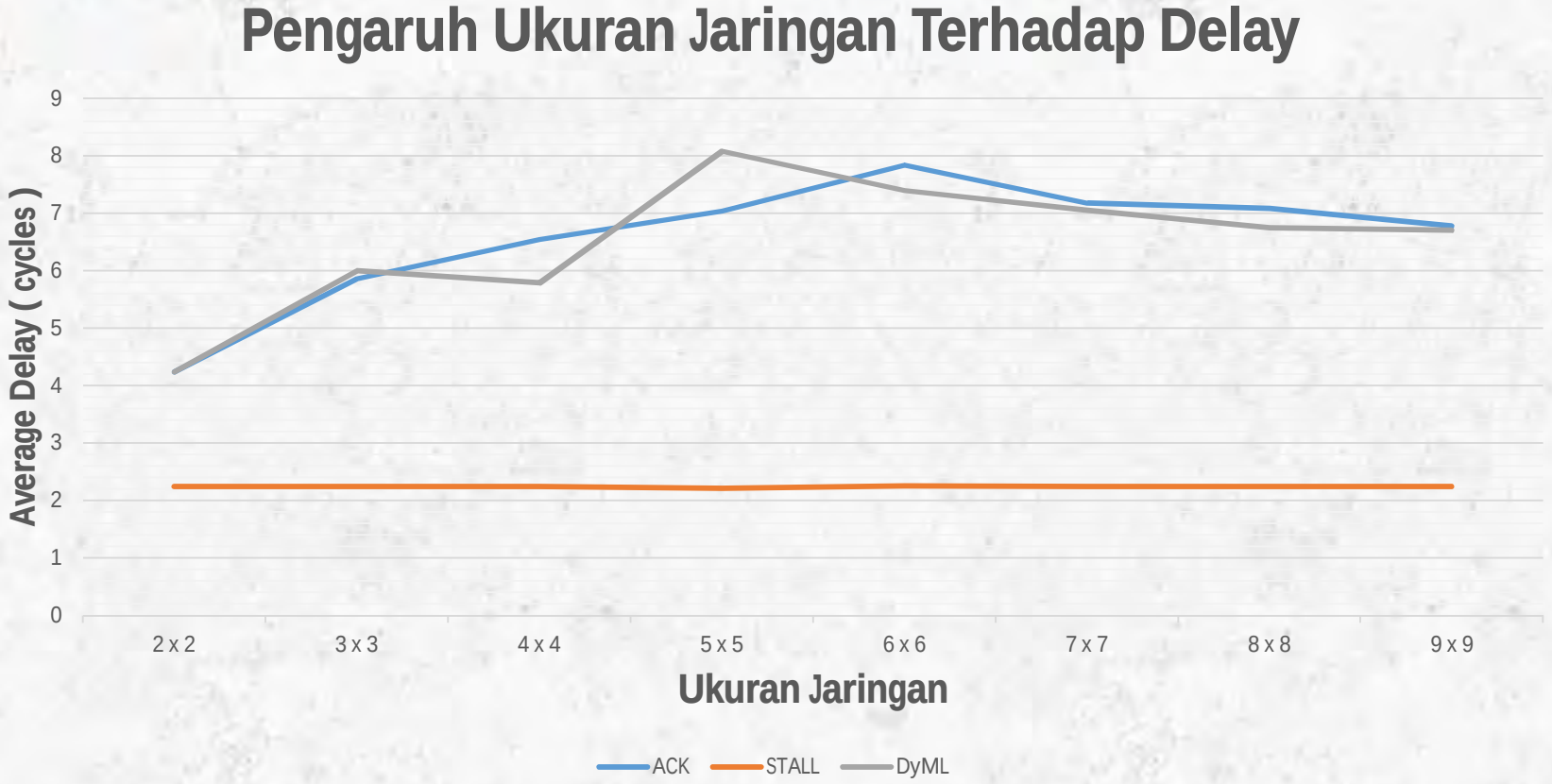
HASIL Model Jaringan 2

Perubahan THROUGHPUT



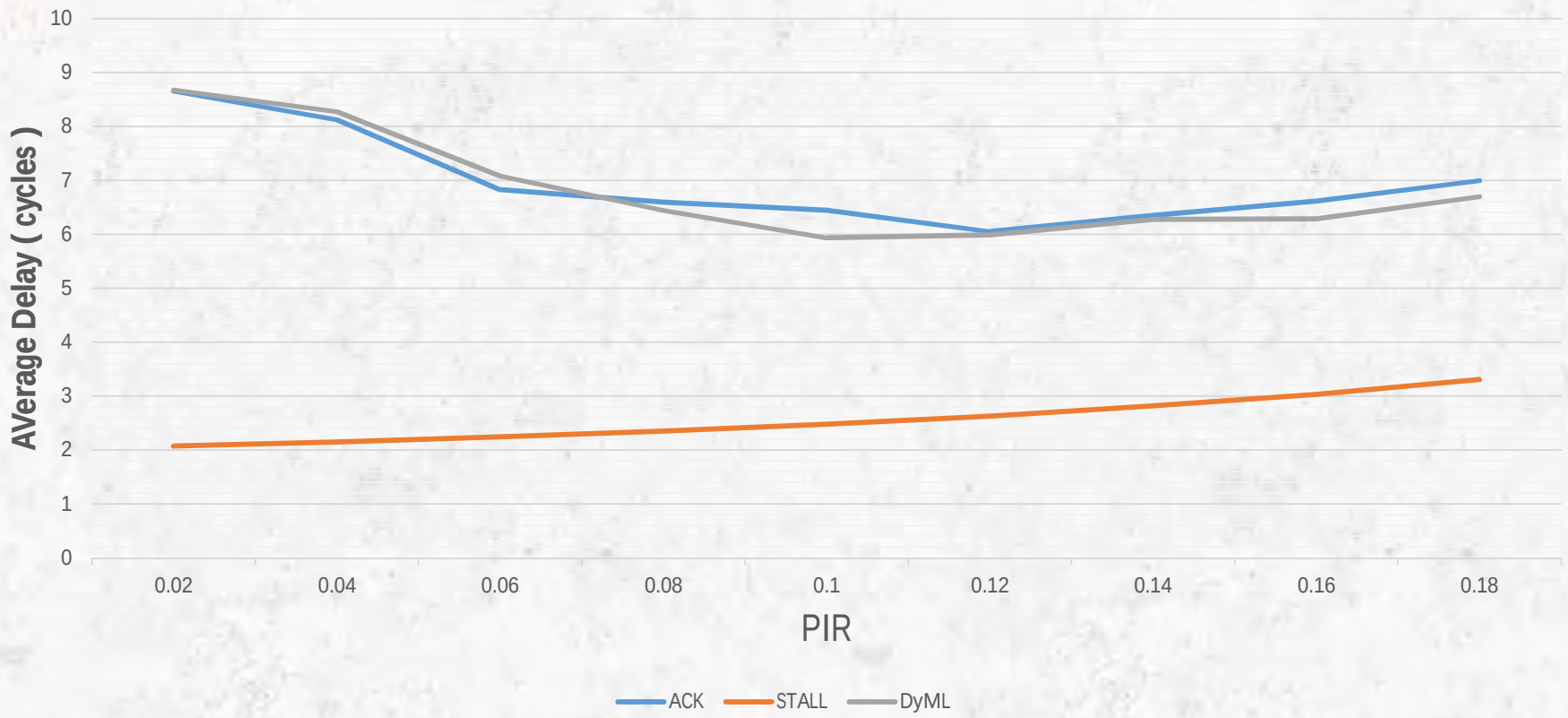
HASIL Model Jaringan 2

Perubahan DELAY



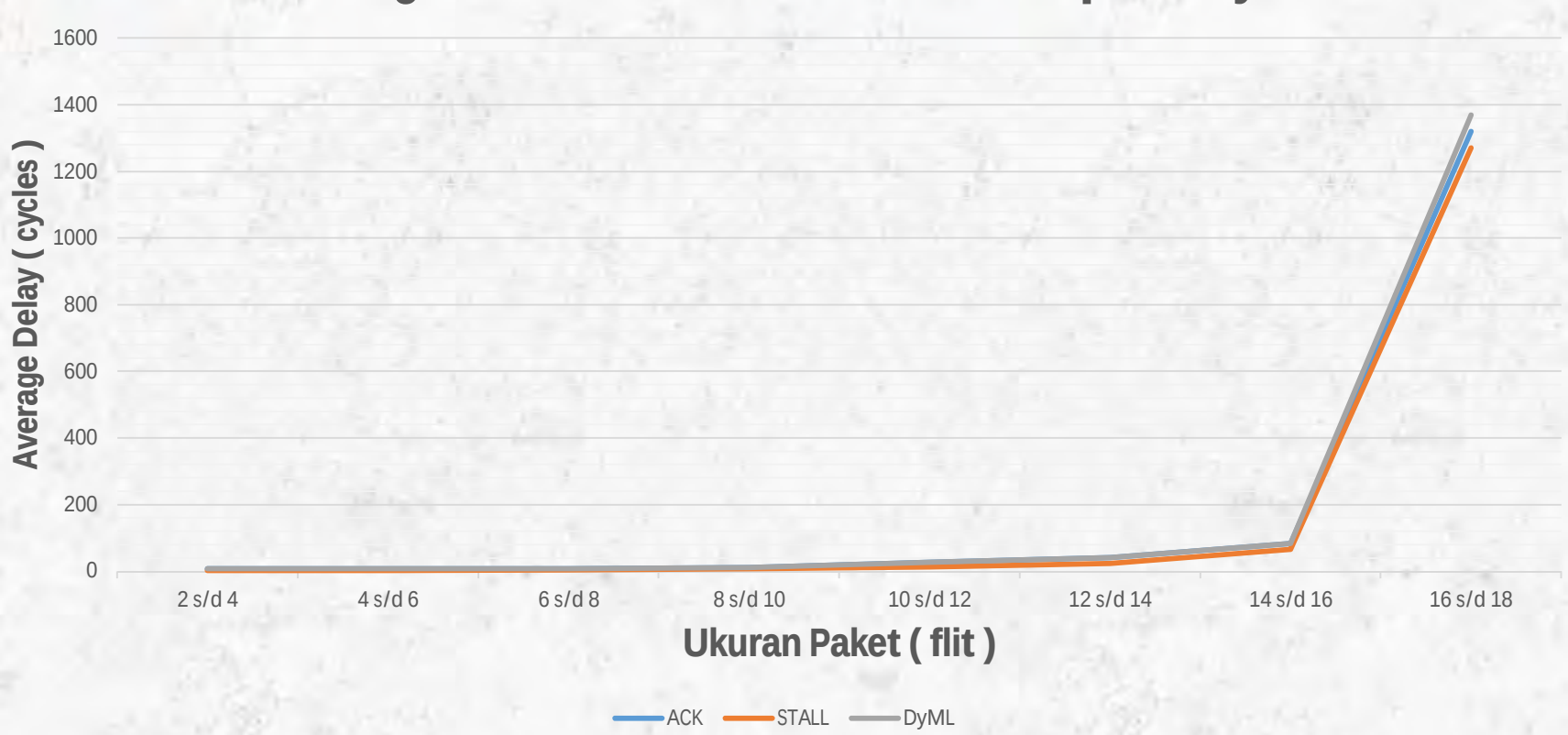
HASIL Model Jaringan 2

Perubahan DELAY



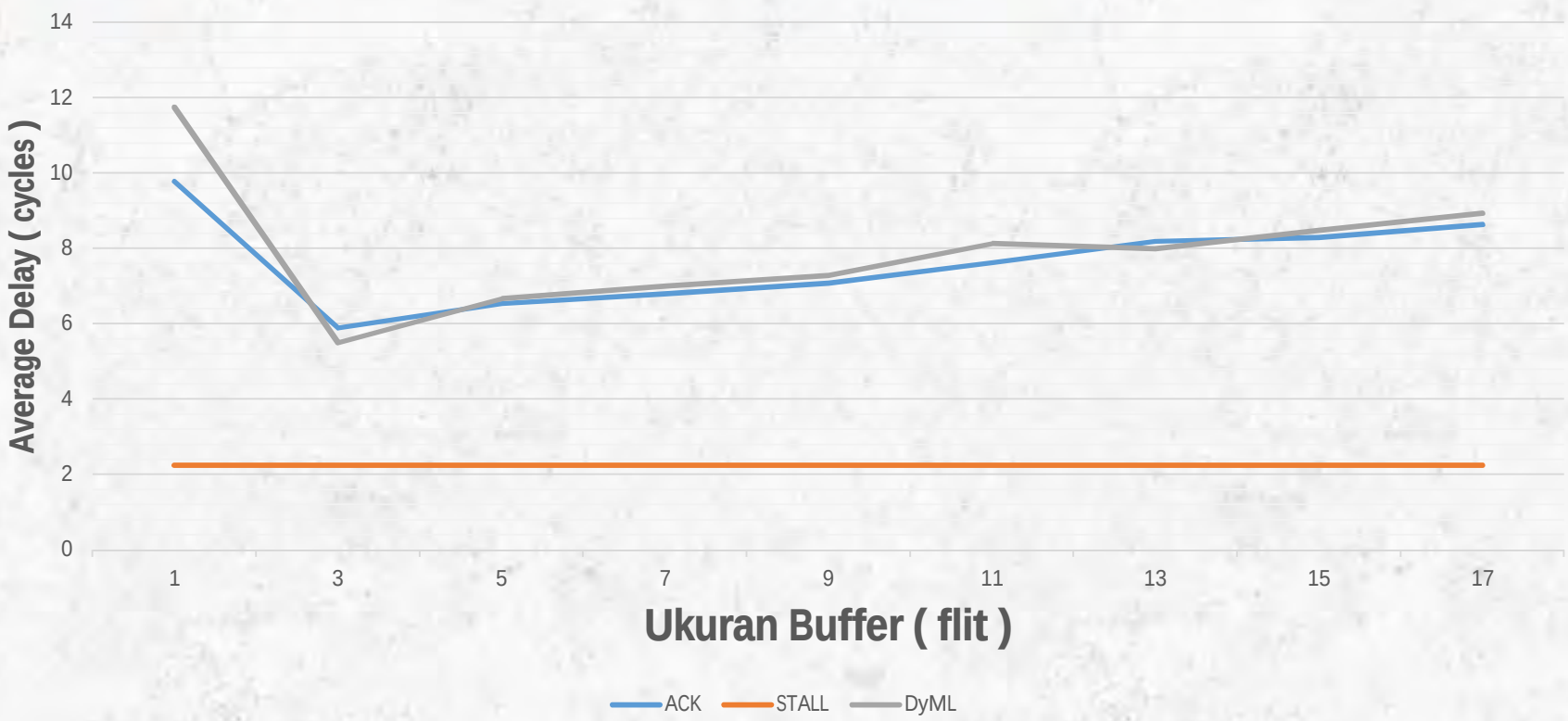
HASIL Model Jaringan 2

Perubahan DELAY



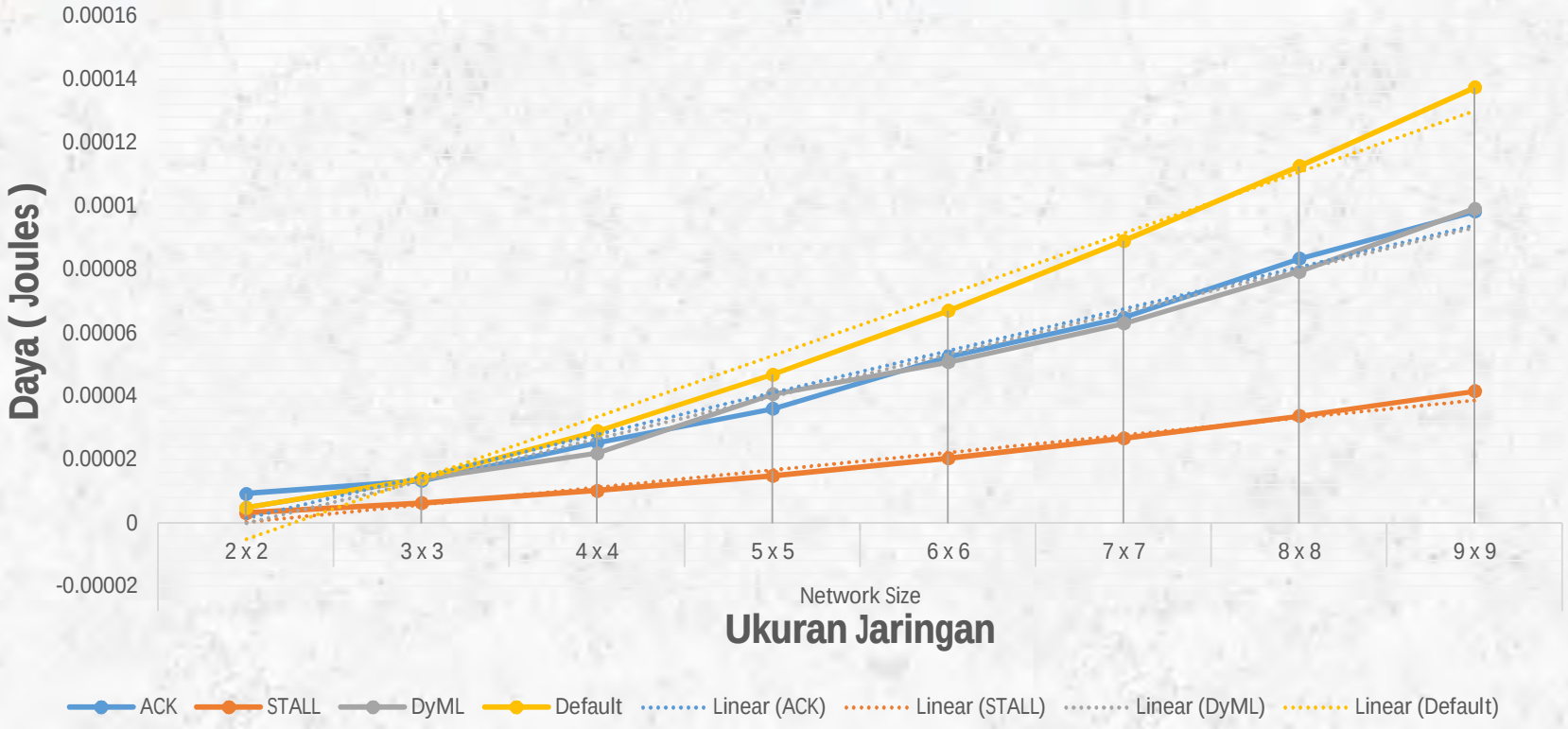
HASIL Model Jaringan 2

Perubahan DELAY



HASIL Model Jaringan 2

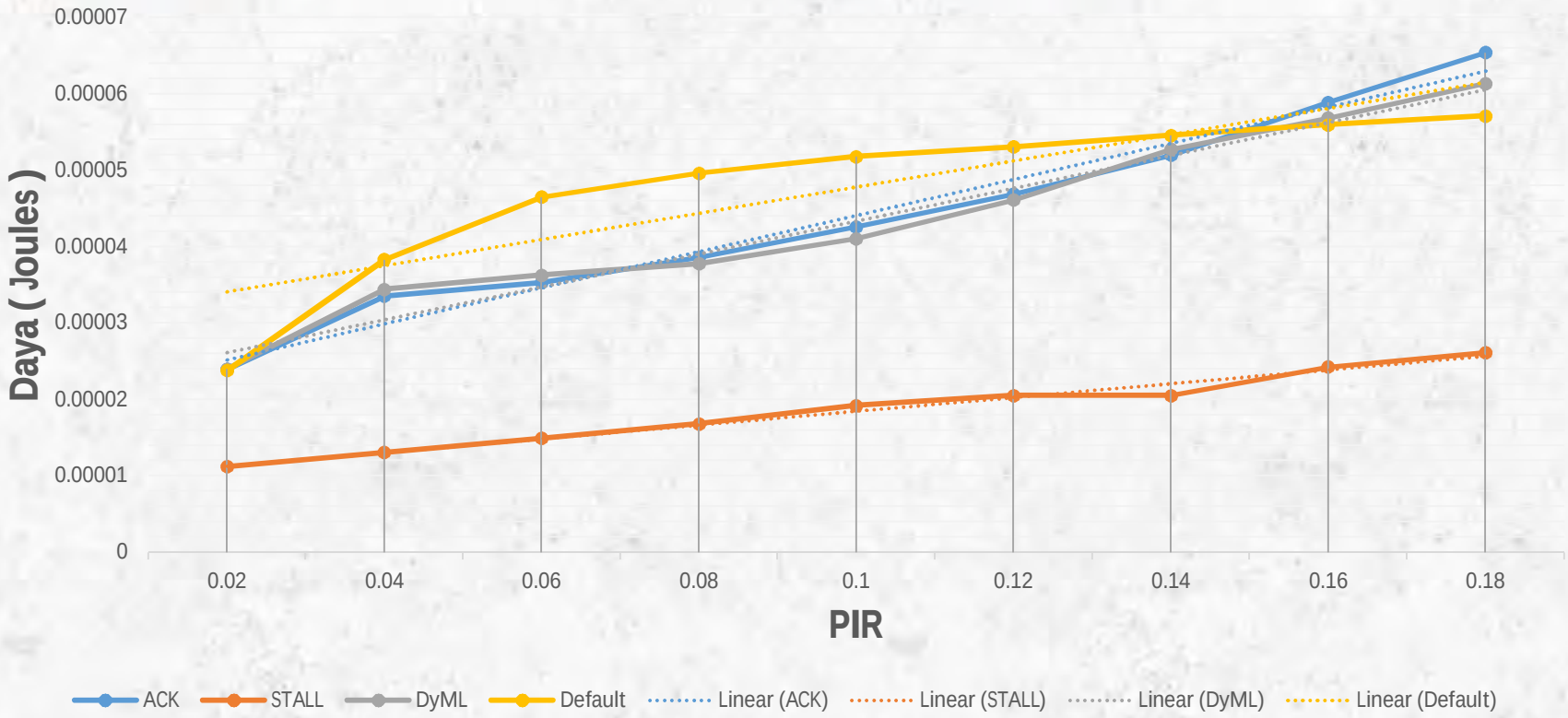
PerubahanDAYA



HASIL Model Jaringan 2

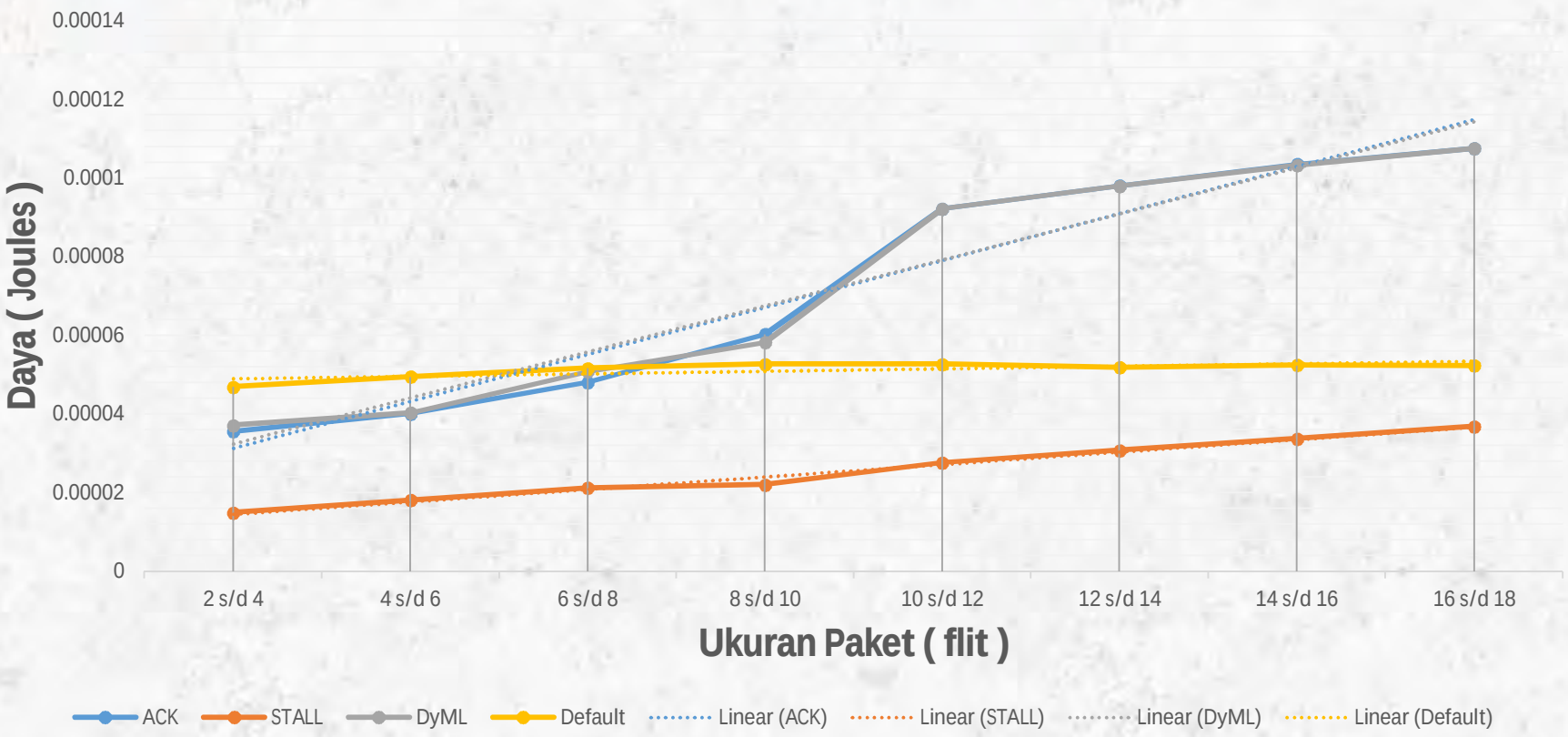
PerubahanDAYA

Pengaruh PIR Terhadap Daya yang Digunakan



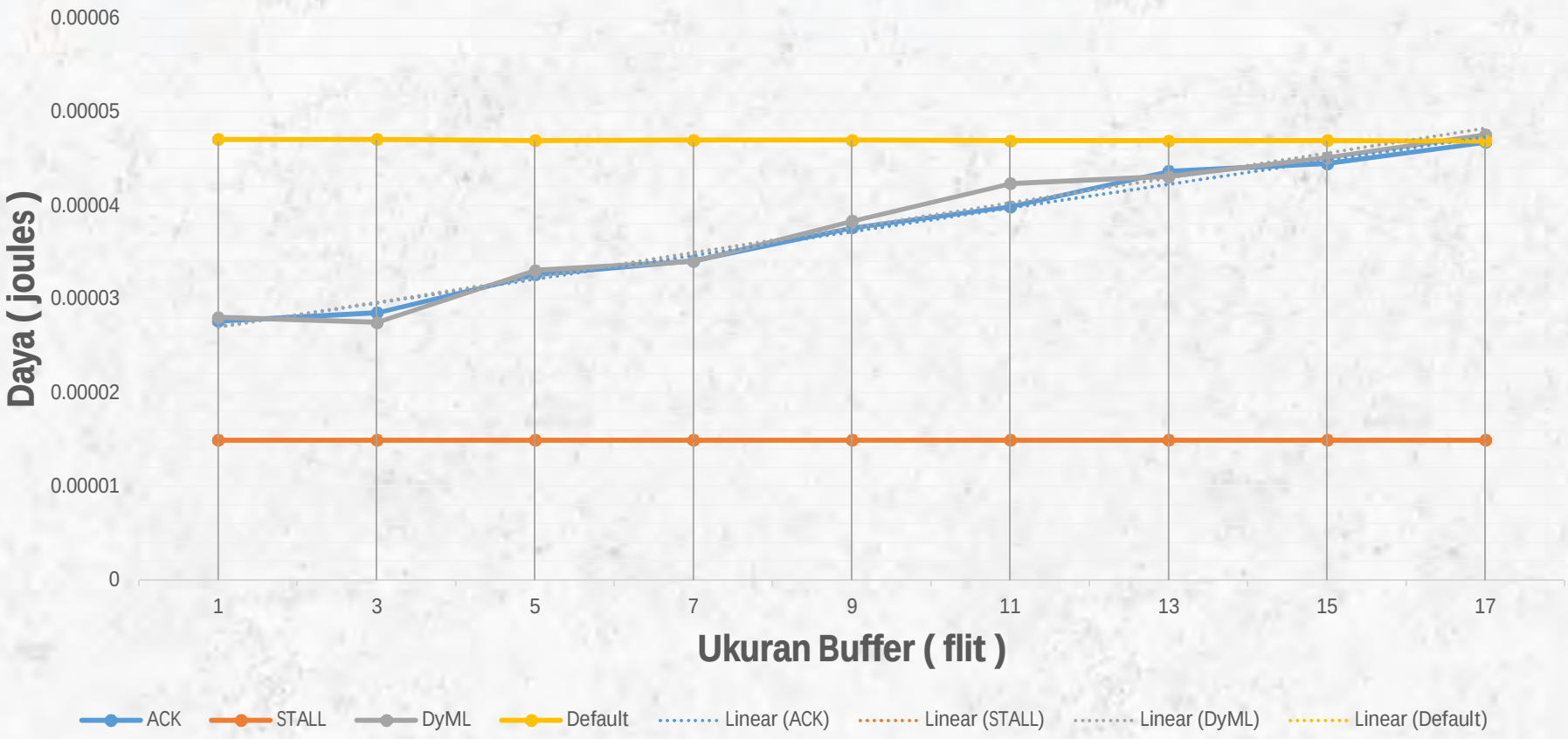
HASIL Model Jaringan 2

PerubahanDAYA



HASIL Model Jaringan 2

Perubahan DAYA



KESIMPULAN

KESIMPULAN

MODEL Jaringan1

- Jaringan tanpa Flow Control mengalami saturasi
- Jaringan dengan Flow Control Stall / Go tidak mengalami saturasi

KESIMPULAN

MODEL Jaringan 2

THROUGHPUT

DELAY & DAYA

STALL

GO

KESIMPULAN

Peningkatan THROUGHPUT

KESIMPULAN

Penurunan DELAY

407 Cycles	606 Cycles	1631 Cycles	322 Cycles
Skenario 1	Skenario 2	Skenario 3	Skenario 4

KESIMPULAN

Penurunan DAYA

TECHNOLOGY SHOULD IMPROVE YOUR LIFE... NOT BECOME YOUR LIFE.

MODEL JARINGAN 1

Parameter Desain

- Topologi : 2D - Mesh
- Ukuran Jaringan : 5 x 5
- Routing : XY Routing
- Trafik : Transpose

MODEL JARINGAN 1

Parameter Desain Skenario 1

- Ukuran Buffer : 2 - 10 flit
- Ukuran Paket : 2 - 4 flit
- Packet Injection Rate : 0.02 - 0.3
- Flow Control : Default, Stall / Go, Ack / Nack , DyML

MODEL JARINGAN 1

Parameter Desain Skenario 2

- Ukuran Buffer : 8 flit
- Ukuran Paket : 2 - 22 flit
- Packet Injection Rate : 0.04 - 0.08
- Flow Control : Default, Stall / Go, Ack / Nack , DyML

MODEL JARINGAN 2

Parameter Desain Skenario 1

- Ukuran Buffer : 8 flit
- Ukuran Paket : 2 - 4 flit
- Packet Injection Rate : 0.06
- Ukuran Jaringan : 2 x 2 - 9 x 9

MODEL JARINGAN 2

Parameter Desain

- Topologi : 2D - Mesh
- Routing : XY Routing
- Trafik : Transpose
- Flow Control : Default, Stall / Go, Ack / Nack , DyML

MODEL JARINGAN

Parameter Desain Skenario 2

- Ukuran Buffer : 8 flit
- Ukuran Paket : 2 - 4 flit
- Packet Injection Rate : 0.02 - 0.18
- Ukuran Jaringan : 5 x 5

MODEL JARINGAN

Parameter Desain Skenario 3

- Ukuran Buffer : 8 flit
- Ukuran Paket : 2 - 18 flit
- Packet Injection Rate : 0.06
- Ukuran Jaringan : 5 x 5

MODEL JARINGAN

Parameter Desain Skenario 4

- Ukuran Buffer : 1 - 17 flit
- Ukuran Paket : 2 - 4 flit
- Packet Injection Rate : 0.06
- Ukuran Jaringan : 5 x 5