Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

BAB II LANDASAN TEORI

2.1 Routing Protocol

Arti *Routing* dalam jaringan adalah suatu proses pengiriman sebuah informasi dalam suatu jaringan dari sebuah sumber sampai dengan tujuannya. Sebuah informasi di *route* sesuai dengan *routing table* yang dimiliki sebuah router. *Routing table* ini biasanya berisikan informasi seperti, *destination*(tujuan akhir dari paket), *Interface*(dari *interface* mana informasi akan dikirim), *Next hop, Metric, Routes*. *Routing table* ini dapat diisi secara manual maupun secara dinamis. [2]

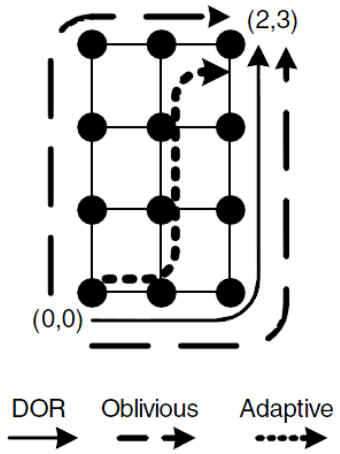
2.1.1 Routing Algoritma Network On Chip (NOC)

Routing algoritma dalam NoC menentukan jalur mana yang pesan akan ikuti dalam jaringan hingga sampai pada tujuannya. Tujuan dari *algoritma routing* adalah untuk mendistribusikan *traffic* secara rata diantara jalur-jalur yang diberikan oleh topologi jaringan, dan juga untuk menghindari *bottlenecks* dan meminimalisir *congestion*, dan meningkatkan *latency* dan *throughput* dari sebuah jaringan seluruh tujuan performa ini harus didapatkan ketika mengikuti aturan yang ketat pada pengimplementasian yang kompleks. Sirkuit pada *routing* dapat meregangkan *critical path delay* dan menambahkan *router's footprint*. Ketika kekurangan energi atau *routing circuity* tergolong rendah, rute yang dipilih dapat mempengaruhi jumlah *hop* langsung, dan mempengaruhi konsumsi energi secara besar [3] [4].

Routing algoritma dibagi menjadi tiga kelas – *deterministic*, *oblivious*, dan adaptive. Meskipun banyak *algoritma routing* yang telah di tawarkan, *algoritma routing* yang biasanya digunakan dalam NoC adalah *deterministic-ordered routing* (DOR), karena kesederhanaannya. DOR adalah contoh dari *algoritma routing deterministic*, dimana semua pesan dari node A ke B selalu melewati jalur yang

Rancang Bangun Algoritma..., Wahyudi Khusnandar, FTI UMN, 2017

sama. Dengan DOR, pesean melintasi jaringan melalui dimensi per dimensi, mencapai koordinat yang sama dengan tujuannya sebelum berpindah ke dimensi yang lain. Gambar 2.1 menunjukkan X-Y DOR mengirim *packet* ke dimensi X dahulu, baru ke dimensi Y. *Packet* berpindah dari node (0,0) akan berpindah 2 lompatan melalui dimensi X, sampai pada node (2,0) sebelum berpindah dari dimensi Y untuk mencapai tujuannya[1].



Gambar 2.1 Tiga Kelas Routing Algoritma

Kelas lain dari *algoritma routing* adalah *oblivious*, dimana pesan dikirim melalui jalur yang beberda dari A ke B, namun jalur dipilih tanpa memperhatikan *network congestion.* Sebagai contoh, *router* dapat secara acak memilih jalur alternatif untuk mengirim sebuah pesan. Gambar 2.1 menunjukkan contoh dimana pesan dari node(0,0) ke node (2,3), dapat secara acak ditentukan dari Y-X route maupun X-Y route. [1]

Routing algoritma yang lebih mutakhir adalah skema *adaptive*, dimana jalur yang akan pesan terima dari titik A ke titik B tergantung dari pada keadaan *congestion* pada jaringan. Sebagai contoh pada Gambar 2.1, pesan mungkin akan memulai dengan mengikuti rute X-Y dan mengalami *congestion* pada *outgoing link*

Rancang Bangun Algoritma..., Wahyudi Khusnandar, FTI UMN, 2017

barat node (1,0). Karena *congestion* ini, pesan akan di kirimkan ke jalur utara menuju tujuan.

Routing Adaptive dapat membatasi untuk meminimalisir jalur antara sumber dan tujuan. Opsi alternatifnya adalah untuk memakai *misrouting*, yang memperbolehkan *packet* untuk menggunakan jalur ke nonproduktif yang menghasilkan jalur yang *non-minimal.* Ketika *misrouting* diperbolehkan, yang menjadi masalah adalah *livelock.* Tanpa mekanisme untuk menjamin kemajuan progres, *livelock* dapat terjadi karena *packet* terus menerus *misroute,* jadi *packet* tidak akan pernah sampai ke tujuannya.

Algoritma *adaptive routing* memerlukan mekanisme untuk melacak *congestion* level pada jaringan, dan memperbarui rute. Banyak mekanisme *congestion sensitive* telah diajukan, dari yang paling sederhana adalah dengan merekam ke informasi yang telah diambil dan digunakan dengan *flow control protocol*. [5]

2.2 Bandwidth, Throughput, Latency, Jitter

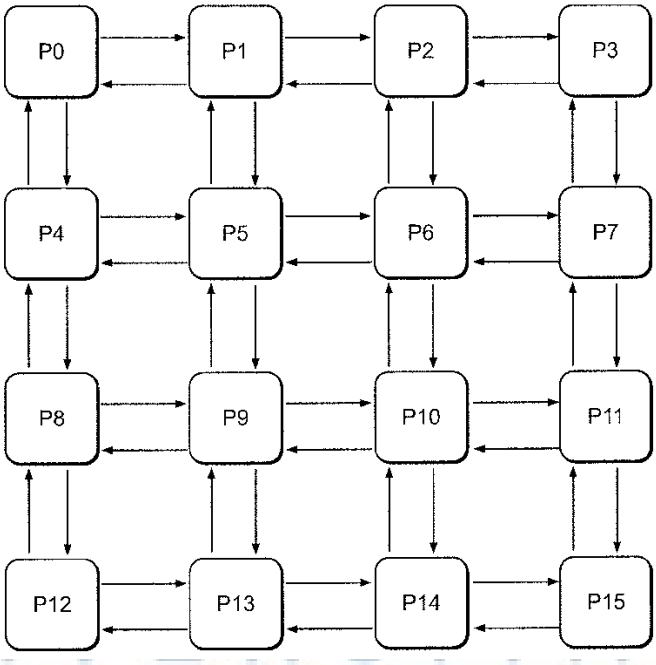
Pada jaringan komputer, *bandwidth* biasa digunakan sebagai sinonim untuk *data transfer rate,* banyak data yang dapat dibawa dari suatu tempat ke tempat lainnya dalam jangka waktu yang telah ditentukan (biasanya dalam detik) [6]. Sedangkan *throughput* adalah banyak data yang berhasil berpindah dari suatu tempat ke tempat lain dalam jangka watu yang telah ditentukan biasanya diukur dalam satuan *bits per second* (bps), *megabits per second* (Mbps), atau *gigabits per second* (Gbps) [7].

Rancang Bangun Algoritma..., Wahyudi Khusnandar, FTI UMN, 2017

Latency adalah istilah yang digunakan untuk mengacu pada waktu yang dibutuhkan bit untuk di kirimkan dari sumber ke destinasi, sedangkan *jitter* adalah delay yang berbeda setiap waktu, ketika beberapa *packets* memerlukan *travel time* yang lebih dari biasanya dari suatu *system* ke *system* lainnya. *Jitter* akan mengakibatkan *congestion, timing drift* dan pergantian rute. [8]

2.3 Konsep Network-on-Chip

Dalam *NoC-based System on Chip,* komponen – komponen dari *chip* dibagi menjadi beberapa *processing blocks*(PB). Satu PB dapat berisikan satu atau grup komponen yang membentuk sebuat *subsystem* dalan PB. Semua PB terkoneksi melalui *router*. Router dapat terkoneksi dengan satu atau lebih dengan *router* tetangga tergantung pada topologi dari jaringan. [1]



Gambar 2.2Contoh NoC Topologi Mesh

NOC menggunakan metode *packet-switched* untuk mengirim data dari satu PB ke PB lainnya. Semua *router* di dalam jaringan memiliki alamat yang digunakan

Rancang Bangun Algoritma..., Wahyudi Khusnandar, FTI UMN, 2017

sebagai pengenal yang unik. Data dan informasi tambahan seperti tujuan *router* dan tipe data di simpan menjadi sebuah flit. Informasi tambahan berada pada header dan biasanya digunakan oleh router untuk menentukan aksi yang akan dilakukan terhadap flit. Setiap PB akan mengirimkan semua flit ke *router* tujuan melalui jaringan. Algoritma *routing* pada setiap *router* akan menentukan semua jalur berikutnya untuk semua flit untuk mencapai tujuannya.

Desain dari NoC dapat dibagi menjadi beberapa aspek, seperti topologi, routing, *flow* control, dan juga *router* microarchitecture. Topologi menentukan bagaimana susunan fisik dan koneksi antar *nodes* dan *channels* dalam jaringan. Desain topologi dimulai dengan keputusan bagaimana *network topology* dasar yang ingin diimplementasikan, sebagai contoh, *tree, ring, star,* atau *mesh*. Optimisasi dan fitur tambahan nantinya diimplementasikan untuk mengembangkan topologi dasar ini.

Algoritma *routing* menentukan jalur yang dilewati dalam jaringan yang akan dilalui pesan hingga mencapai tujuan. Kemampuan algoritma *routing* untuk menyeimbangkan *traffic* memiliki dampak langsung pada *throughput* dan performa jaringan. *Flow control* menentukan bagaimana suatu sumber daya dialokasikan untuk pesan seiring pesan tersebut berjalan dalam jaringan. Tantangan lainnya datang dari aspek desain komunikasi, seperti *routing algortihm, congestion control, energi and power management*.

Pemodelan aplikasi dan optimisasi dari sebuah aplikasi adalah tantangan lainnya dalam NoC berbasis MPSoC. Setiap aplikasi memiliki pola komunikasi

Rancang Bangun Algoritma..., Wahyudi Khusnandar, FTI UMN, 2017

trafik yang berbeda, bahkan untuk aplikasi yang sama, perbedaan *application mapping* akan mempengaruhi traffic dari jaringan secara signifikan.

2.4 Router Microarchitecture

Fungsi utama *router* adalah untuk meneruskan setiap flit yang tiba pada salah satu *input* channel ke *output channel* yang sesuai. Beberapa tahapan harus diselesaikan sebelum flit dapat pindah ke *output port* yang diinginkan. Pertama adalah *virtual channel (VC) Allocator*. Ketika flit sampai pada *input buffer* pada tiap port pada router, *VC Allocator* akan memberikan *grant* pada salah satu *virtual channel* yang akan di *routed.* [1]

Berikutnya adalah komputasi rute. *Routing logic* di dalam *router* akan memilih *output port* yang cocok sesuai dengan *algoritma routing*. Saat flit dimasukan kedalam *output port*, flit dapat berpartisipasi dalam *switch allocation. Switch allocator* bertugas untuk membuat *crossbar schedule* dengan memasukan *time slot* untuk menunggu flit pada *input* port *router.* Pada intinya *switch allocator* harus menyelesaikan konflik antara flit yang ditujukan pada *output* port yang sama [1]

Akhirnya, setelah menerima *grant* dari *switch* allocator, flot dapat dikirim ke *central crossbar* switch pada *router* pada cycle berikutnya untuk sampai pada destinasi output dan melanjutkan perjalanan ke jaringan. *Allocator* mempresentasikan aspek penting dalam *router design*, karena *allocator* secara langsung mempengaruhi performa pada jaringan dalam beberapa cara. [1]

2.3.1 UTAR NoC Router Overview

UTAR NoC terdiri dari beberapa UTAR NoC router, dan banyak *router* tergantung dari ukuran dari jaringan itu sendiri. Gambar 2.3 menggambarkan

Rancang Bangun Algoritma..., Wahyudi Khusnandar, FTI UMN, 2017

gambaran luas dari UTAR NoC *router*. UTAR NoC *router* memiliki 5 koneksi *port* yaitu *North, East, South, West,* dan juga *PB* . Ada 8 *pin* pada setiap *port* pada *router*. [1]

flit_in pin ini digunakan untuk menerima flit dari *router* tetangga maupun *PB* ke router

EN_flit_in pin ini digunakan untuk memberi sinyal *enable* datangnya *flit* ke *router.*

flit_out pin ini digunakan untuk mengeluarkan *flit* dari *router* ke *router* tetangga.

EN_flit_out pin ini digunakan untuk memberi sinyal *enable* keluarnya *flit* ke *router* selanjutnya.

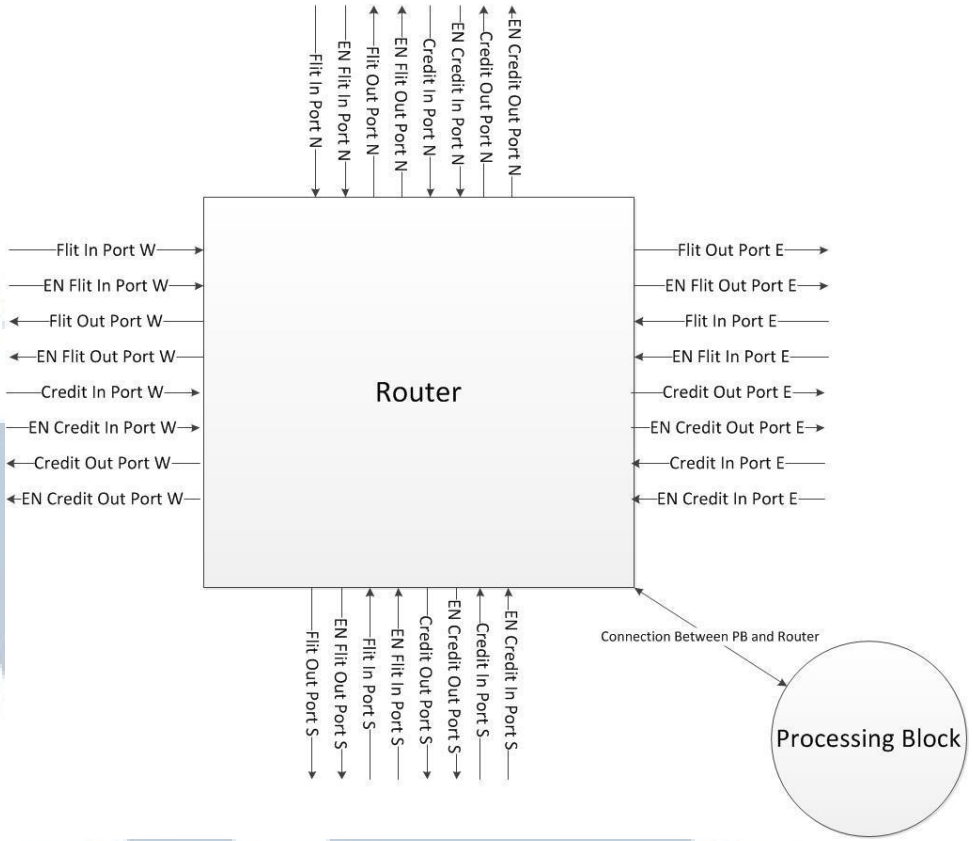
credit_in pin ini digunakan untuk menerima *credit* masuk dari *router tetangga* atau *PB* ke *router.*

EN_credit_in pin ini digunakan untuk memberi sinyal *enable* datangnya *credit* ke *router.*

credit_out pin ini digunakan untuk mengeluarkan *credit* dari *router* ke *router* tetangga.

EN_credit_out pin ini digunakan untuk memberi sinyal *enable* keluarnya *credit* ke *router.*

UNIVERSITAS MULTIMEDIA NUSANTARA



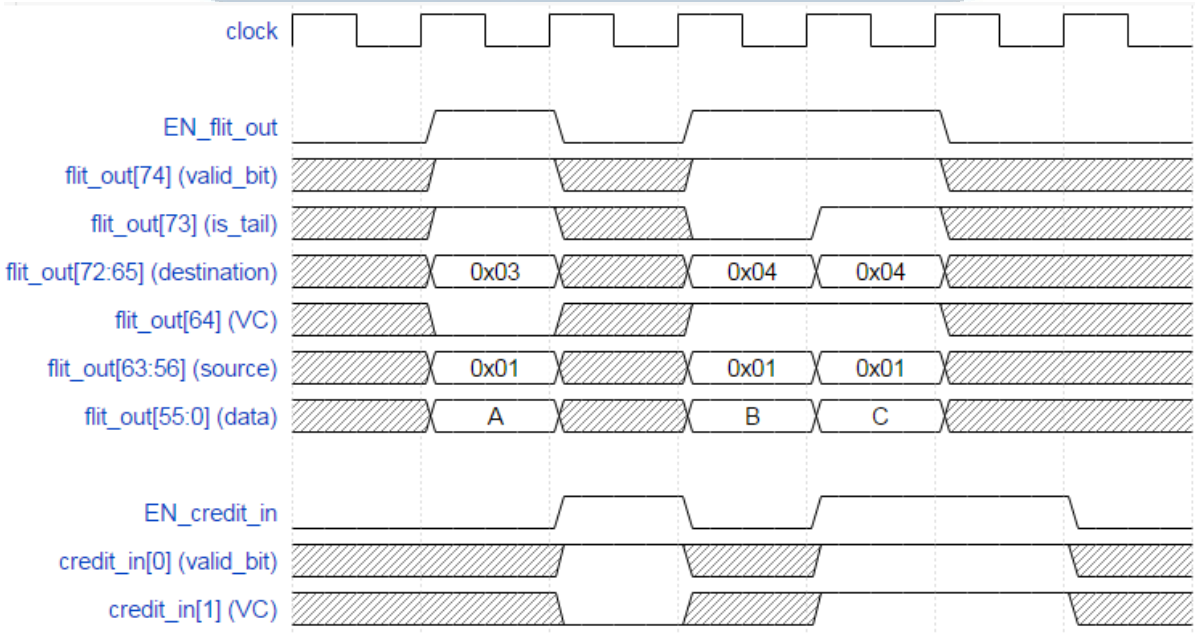
Gambar 2.3 Overview UTAR NoC Router

Pada penelitian UTAR NoC digunakan 75 bits untuk *flit_in* dan juga *flit_out* pin, 2 bit untuk *kredit_in* dan *kredit_out*, dan 1 bit untuk semua sinyal *enable*. *Flits* dasar dari *router* dapat dilihat pada Gambar 2.4 "*valid_bit*" digunakan sebagai penanda validasi dari *flit*. Nilai bit ini akan bernilai satu ketika mengirim *flit* baru. "is_tail" digunakan sebagai penanda *flit* tersebut adalah *flit* akhir atau bukan *flit* akhir. *Packet* besar dapat dipotong menjadi beberapa bagian *flit*, dan *flit* terakhir adalah *flit* yang membawa bit "is_tail" bernilai satu. Destinasi memberikan informasi dari alamat destinasi flit. Lebar dari bit ini tergantung dari ukuran jaringan. *Virtual channel* menandakan VC mana yang akan digunakan untuk mengirim *flit*. Bit Source memberikan informasi dari alamat sumber flit. Bit terakhir adalah bit data. [1]

Tabel 2.1 UTAR NoC Flit

valid_bit	is_tail	destination	virtual channel	source	data
1-bit	1-bit	8-bit	1-bit	8-bit	56-bit

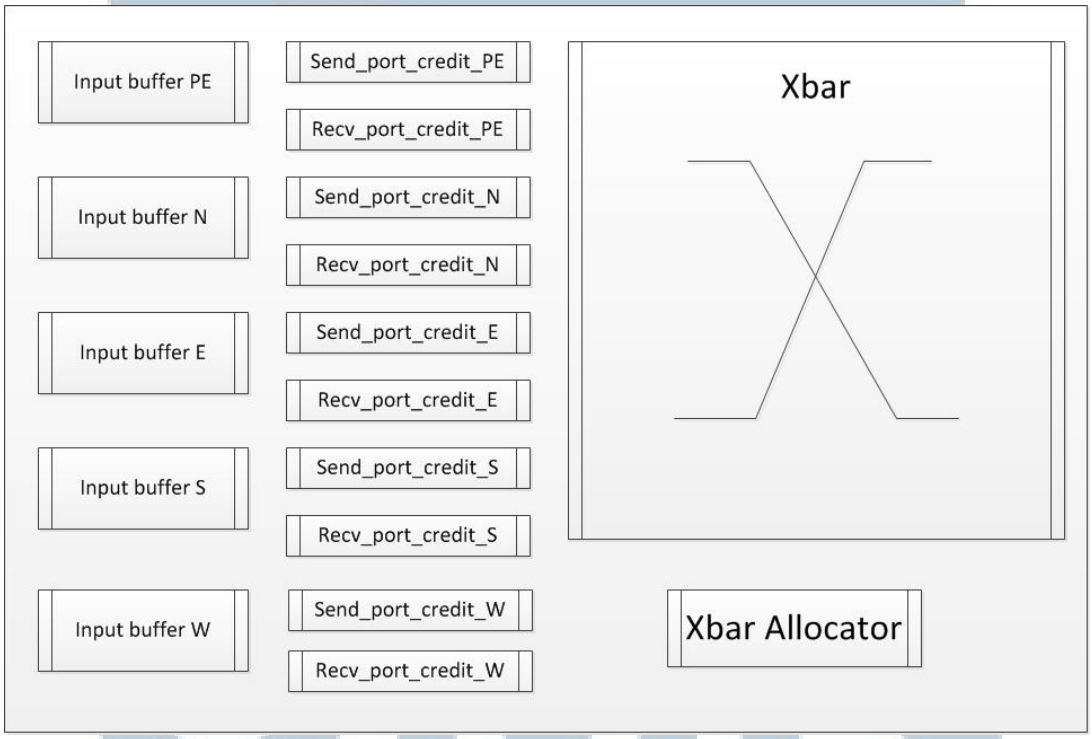
Sinyal yang terkoneksi antar setiap router dapat dijadikan menjadi 2 kategori port, yaitu port *send* dan port *receive*. Port *send* digunakan untuk mengirim *flit* dan menerima *kredit*, sedangkan port *receive* digunakan untuk menerima *flit* dan mengirim *kredit*,. Gambar 2.5 menunjukkan *timing diagram* dari router ketika mengirim flit dan menerima kredit dari port *send*. Nilai pin EN_flit_out diubah menjadi satu setiap *router* mengirim keluar sebuah *flit.* Nilai Bit valid_bit perlu diubah menjadi satu ketika nilai pin EN_flit_out diubah menjadi satu untuk mengindikasikan bahwa flit yang dikirim adalah valid seperti yang bisa dilihat pada *timing diagram* pada clock cycle kedua.



Gambar 2.4 Pengiriman flit dan penerimaan credit timing diagram

Router akan mengirimkan *flit* melalui jaringan menggunakan VC 0 dengan mengubah nilai pada bit ke 64 flit_out yaitu bit Vc menjadi nol. Pada clock cycle ke empat dan ke lima *router* mengirim dua flit dengan destinasi yang berbeda dari flit pertama, dan mengubah nilai bit VC menjadi satu. Flit tersebut dapat diangap

sebagai flit akhir apabila pada bit ke 73 flit_out yaitu bit is_tail bernilai satu. *Router* yang akan menerima akan mengirim banyak *kredit* sesuai dengan banyak *flit* yang diterima. Nilai EN_credit_in akan diubah menjadi satu oleh *router* penerima flit, untuk mengindikasikan bahwa akan ada *kredit* yang akan datang. Pada clock cycle ketiga pada *timing diagram* pada Gambar 2.5 *router* penerima mengirim *kredit* dengan mengubah nilai pin credit_in menjadi satu. Kredit tersebut mengindikasikan bahwa flit yang telah dikirim telah meninggalkan buffer pada *router* penerima.



Gambar 2.5 UTAR Router Microarchitecture

Pada Gambar 2.6 menunjukkan mikroarsitektur dari UTAR NoC router. Router memilik 5 *input port* dan *output port* yang berhubungan dengan 4 dan sebuah port *processing element* (PE) lokal. Komponen inti dari sebuah *router* adalah *input buffer, route computation logic, virtual channel allocator, switch allocator, credit mechanism,* dan *crossbar switch*. Setiap *packet* di simpan dalam

Rancang Bangun Algoritma..., Wahyudi Khusnandar, FTI UMN, 2017

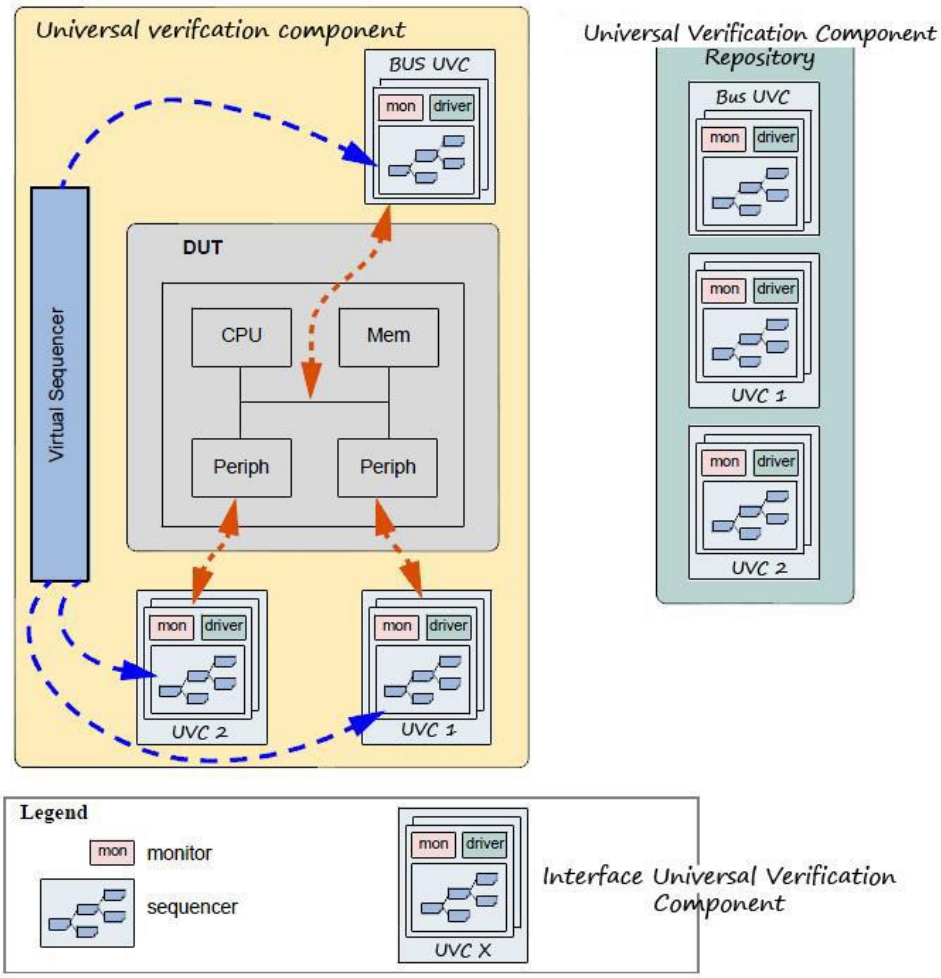
buffer hanya pada *input* ports. Buffer bertugas untuk menyimpan *flit* ketika *flit* masuk ke *router*, dan disimpan selama durasi *router* berjalan. *Route computation logic* menentukan semua jalur dari sumber *flit* ke tujuan *flit*. *Allocator* (VC dan crossbar) menentukan flit mana yang terpilih untuk melanjutkan ke tahap berikutnya dimana mereka melewati crossbar. Mekanisme credit menjalankan perannya dalan *flow control*. Pada akhirnya tugas crossbar switch untuk memindahkan flit dari input port ke output port.

2.5 Universal Verification Methodology [1]

Universal Verification Methodology (UVM) adalah metodologi standar untuk menguji *integrated circuit design.* UVM adalah turunan dari OVM *(Open Verification Methodology).* UVM dibuat berdasarkan SystemC dan *System Verilog Object-Oriented Programming* (OOP). Komponen UVM menggunakan *transaction level modelling* (TLM) untuk komunikasi antara objek. UVM berbasis *testbench* yang terorganisasi secara hierarki. *Testbench* UVM terdiri atas *reusable verification environment* disebut *Unviersal Verification Components (UVC)*. UVC terenkapsulasi, siap untuk digunakan, *verification environment* yang dapat dikonfigurasi untuk *interface protocol*, *submodule* desain, atau *full system* [1].

Setiap UVC mengikuti arsitektur yang konsisten dan terdiri dari kumpulan lengkap dari elemen untuk menstimulasi, memeriksa, dan mengumpulkan informasi untuk protokol khusus ataupun desain. Kompnonen *verification* di aplikasikan pada *device under test* (DUT) untuk memverifikasikan implementasi sebuah protokol ataupun arsitektur desain. Gambar 2.7 menunjukkan contoh dari *verification environment* dengan tiga interface UVC [1].

Rancang Bangun Algoritma..., Wahyudi Khusnandar, FTI UMN, 2017



Gambar 2.6 Contoh Verification Environment

2.6 MCSL NoC Traffic Pattern Suite

Mengintegrasikan beberapa *processing units* pada satu *chip*, dapat menghasilkan performa lebih besar pada setiap energi dan mengurangi *cost* setiap fungsi pada aplikasi dengan *burgeoning complexity*. Performa pada MPSoC tidak hanya ditentukan oleh performa setiap *processing unit*, namun juga bagaimana efisiensi mereka berkolaborasi bersama.

NoC *traffic pattern* yang berperan sebagai *benchmark program*, adalah apliaksi penting untuk mengevaluasi suatu performa dalam NoC dan mengeksplorasi arsitektur [7]. Beberapa pekerjaan mempelajari properti dari *on-*

chip traffic dan mengajukan beberapa *traffic models* yang membuat pola seperti *traffics.* Grecu dan kawan kawan menyediakan garis besar dari kebutuhan untuk NoC *benchmarks* [8]. Soteriou dan kawan kawan mengajukan *3-tuple traffic mode* untuk menangkap karakteristik dari *on-chip traffic* secara empiris [9].

Sekarang ini sudah banyak NoC *simulation environments* yang menyediakan *traffic pattern* yang acak untuk pembelajaran NoC. Pendekatan seperti penggunaan *problability distribution* untuk mengacak karakteristik *on-chip communication traffic characteristics* [7]. Ketika telah dikonfigurasi secara tepat, *traffic traffic* ini dapat mengukur rata-rata performa NoC dan *power consumption* secara jangka panjang. *Traffic pattern* yang realistik dapat memberikan hasil performa dan *power consumption* yang lebih dan memberikan informasi secara detail untuk meningkatkan arsitektur NoC.

MCSL NoC *traffic suite* adalah salahsatu *traffic benchmark* yang realistik. Saat ini MCSL NoC *traffic suite* menyediakan 8 realistik *traffic pattern*, yang berisikan aplikasi MPSOC yang biasanya digunakan dan termasuk arsitektur NoC yang populer dalam berbagai skala. MCSL tidak ganya menangkap *communication behaviors* dalam NoC namun ketergantungan *temporal* diantara mereka. Setiap *traffic pattern* dalam MCSL memiliki dua versi, RTP (*recorded traffic pattern*) dan STP (*statistical traffic pattern*).

UNIVERSITAS
MULTIMEDIA
NUSANTARA