Faktor Exacta *11 (3):* 275-290, 2018
p-ISSN: 1979-276X
e- ISSN: 2502-339X
DOI : 10.30998/faktorexacta.v11i3.2784

SIMULATOR ARSITEKTURAL DARI SIRKUIT ELEKTRONIS GUNA TUJUAN PEMBELAJARAN

SURYO BRAMASTO
SUNARTO

Program Studi Informatika
Institut Teknologi Indonesia
Jalan Raya Puspiptek Serpong, Kota Tangerang Selatan, Banten-15320
Nomor telephone: +62217561102, +6281563470055
E-mail: suryo.bramasto@iti.ac.id, sunarto@iti.ac.id

Abstrak. Proses pembelajaran rancang bangun sirkuit elektronis pada umumnya membutuhkan infrastruktur-infrastruktur spesifik yang terkadang menjadi kendala dalam pengadaannya dikarenakan faktor biaya. Simulator sirkuit dapat menjadi alternatif sebagai alat bantu pembelajaran proses rancang bangun sirkuit elektronis. Simulasi arsitektural memungkinkan desainer untuk memeriksa berbagai pilihan desain. Dewasa ini, desain sistem yang semakin cenderung menuju arah arsitektur yang beraksi terhadap fenomena tingkat sirkuit secara kemampuan telah mengungguli kapabilitas simulator-simulator arsitektural berbasis daur. Pada artikel ini disajikan sebuah simulator yang mencakup kapabilitas pemodelan sirkuit dan memungkinkan simulasi tingkat arsitektural yang bereaksi terhadap karakteristik sirkuit (seperti latensi, energi, kecerahan daya, atau seri arus) pada basis daur per daur. Simulasi arsitektural yang reaktif terhadap kerakteristik sirkuit memungkinkan variasi pada kecepatan simulasi. Guna kepentingan pembelajaran, tersedia *template* berbagai macam sirkuit elektronis baik analog maupun digital, serta juga contoh-contoh sirkuit terbangun. Lingkungan terbangun memungkinkan visualisasi proses serta simulasi sirkuit analog dan digital. Sistem simulator memungkinkan berbagai variasi pelatihan skala laboratorium, sehingga memungkinkan siswa mengikuti karakteristik proses pada sirkuit analog dan digital secara visual. Pada penelitian ini diterapkan metodologi rancang bangun piranti lunak yakni *Rational Unified Process* (RUP) dan paradigma pemrograman berorientasi objek. Dilakukan juga optimasi performa pada simulator dengan *benchmarking* terhadap SimpleScalar.

Kata kunci: OOP, optimasi performa, pembelajaran, RUP, simulasi arsitektural, simulasi sirkuit

Abstract. *The electronic circuit engineering learning processes generally requires specific infrastructures which sometimes constrained to cost factors in the procurement. Circuit simulator could be an alternative as electronic circuit engineering learning tool. Architectural simulation provides designers the ability to quickly examine a wide variety of design choices. The recent trend in system design toward architectures that react to circuit-level phenomena has outstripped the capabilities of traditional cycle-based architectural simulators. In this paper, a simulator that incorporates a circuit modeling capability, permitting architectural-level simulations that react to circuit characteristics (such as latency, energy, power brightness, or current draw) on a cycle-by-cycle basis is presented. As for learning purpose, circuit template and pre-built circuits for many categories are provided. The environment enables process visualization and simulation of analog and digital circuits. The system enables the*

creation of many laboratory exercises, which offer students opportunities to follow visually characteristic processes in analog and digital circuits. At this research, Rational Unified Process (RUP) software process model and Object Oriented Programming (OOP) are implemented.

Keywords: *architectural simulation, circuit simulation, learning, RUP, OOP*

PENDAHULUAN

Proses pembelajaran rancang bangun sirkuit elektronis pada umumnya membutuhkan infrastruktur-infrastruktur spesifik yang terkadang menjadi kendala dalam pengadaannya dikarenakan faktor biaya. Simulator sirkuit dapat menjadi alternatif sebagai alat bantu pembelajaran proses rancang bangun sirkuit elektronis. Simulator arsitektural biasa dimanfaatkan oleh arsitek perangkat keras guna mempercepat daur desain. Namun, secara spesifik simulator sirkuit elektronis analog dan digital dapat juga digunakan sebagai alat bantu belajar misalnya pembelajaran Complementary Metal-Oxide Semiconductor (CMOS), pembelajaran logika transistor-transistor (TTL) atau pembelajaran pemrograman Peripheral Interface Controller (PIC). Simulator arsitektural pada awalnya diimplementasikan secara langsung dengan bahasa pemrograman atau *hardware design language* (HDL) guna eksekusi dan validasi ketepatan desain perangkat keras. Secara performa, model piranti lunak berjalan lebih lambat dibanding perangkat keras yang dimodelkan, tetapi model piranti lunak dapat dibangun dengan cepat dibandingkan dengan perangkat keras yang dimodelkan, yang mana membutuhkan proses fabrikasi. Dengan demikian model perangkat lunak yang dapat dibangun lebih cepat akan mempersingkat daur desain sehingga memungkinkan arsitek perangkat keras mengevaluasi lebih banyak desain sebelum penentuan guna proses fabrikasi.

Pendekatan tradisional pada simulator arsitektural dipenuhi dengan terlebih dahulu membangun model perangkat lunak dari arsitektur perangkat keras atau sirkuit elektronis, mengidentifikasi komponen-komponen utama dari perangkat keras atau sirkuit elektronis, dan menentukan latensi operasi sebagai fungsi daur *clock* yang diharapkan dari mesin. Sebagai contoh Arithmetic Logic Unit (ALU) atau register memiliki satu daur, sedangkan latensi dari *cache* tergantung dari ukuran *cache* tersebut. Setelah komponen-komponen latensi terdefinisi, sebuah model arsitektural lengkap dari suatu mesin dibangun dengan menghitung jumlah tiap komponen, membangun koneksi-koneksi dalam arsitektur mikro, dan menentukan resiko kegagalan (*hazards*/*stall*) yang dapat terjadi pada instruksi-instruksi terkait komponen-komponen yang ditentukan pada desain. Setelah model ditentukan, simulasi arsitektural menjadi proses penentuan jumlah daur keseluruhan yang dibutuhkan saat mengeksekusi sebuah program dengan memperhitungkan eksekusi instruksi-instruksi pada komponen spesifik, sumber daya tersedia, serta kegagalan-kegagalan yang terjadi saat eksekusi instruksi.

Dewasa ini dimungkinkan untuk mengambangkan desain arsitektur komputer yang dapat beradaptasi terhadap fenomena tingkat sirkuit. Pada sistem yang adaptif tersebut, dimungkinkan bagi arsitektur komputer untuk mempengaruhi operasi sirkuit dan sebaliknya. Penerapan sistem adaptif tersebut antara lain pada penelitian yang dilakukan oleh Kevin Skadron dan kawan-kawan (2017) yakni *thermal throttling*, serta penelitian oleh D. Ernst dan kawan-kawan (2003) yakni *Razor clocking*. Optimasi pada arsitektur-arsitektur *circuit-aware* tersebut memiliki kebutuhan yang kurang lebih sama dengan simulator arsitektural yang harus secara akurat membaca detil fenomena sirkuit guna mensimulasikan operasi mesin yang sedang dipelajari dengan benar. Detil tersebut misalnya perhitungan jumlah total *switching* antar perangkat pada setiap daur operasi implementasi sirkuit, atau detil informasi pewaktuan dari *pipeline stages* pada

basis per daur. Kebutuhan akurasi pembacaan detil tersebut dipenuhi oleh model-model sirkuit *analytical* hingga tingkat komponen-komponen *microarchitecture*. Keunggulan utama dari model-model sirkuit *analytical* adalah kecepatan dan fleksibilitas. Walau demikian akurasi dari model-model sirkuit *analytical* yang disederhanakan seringkali menjadi pertanyaan.

Pada artikel ini disajikan infrastruktur pemodelan simulasi arsitektural guna simulasi sirkuit elektronis. Simulasi sirkuit yang disajikan memiliki akurasi yang cukup karena mencakup fenomena sirkuit secara detail termasuk delay gerbang individual dan karakteristik energi. Performa simulasi dipenuhi dengan optimasi kombinasi simulasi sirkuit dengan simulasi tingkat arsitektural. Optimasi yang diimplementasikan mencakup *circuit timing memoization* dan *fine-grained instruction sampling*.

Selebihnya dari artikel ini disusun sebagai berikut, bagian "Latar Belakang dan Penelitian Terkait" memaparkan detail penelitian terkait tentang simulasi arsitektural, simulasi sirkuit, dan optimasi performa simulator; bagian "Metodologi" memaparkan detail metodologi simulasi sirkuit dan integrasinya ke model simulasi arsitektural, serta metodologi rancang bangun simulator sirkuit; bagian "Optimasi Performa" menjelaskan implementasi optimasi guna meningkatkan performa simulator; bagian "Contoh Studi Kasus" menunjukkan simulator untuk pembelajaran pembelajaran CMOS, TTL, dan PIC. Sedangkan bagian terakhir merupakan penutup yang berisi kesimpulan dan saran.

Latar Belakang dan Penelitian Terkait

T. Austin dan kawan-kawan (2002) menyatakan bahwa infrastruktur simulasi arsitektural yang ada dan cukup populer di kalangan akademisi mupun industri yakni SimpleScalar. SimpleScalar merupakan simulator arsitektur komputer, dimana model paling detail untuk SimpleScalar mampu mengeksekusi program dengan laju 100.000 instruksi per detik (IPS). Infrastruktur simulasi arsitektural berikutnya yang juga menjadi acuan pengembangan simulator sirkuit pada artikel ini yakni Cai Lim power-performance simulator oleh George Z. N. Cai dan kawan-kawan (2012), yang memperkenalkan model-model kerapatan daya dari desain data internal Intel® untuk blok-blok mikroarsitektural fungsional. Cai Lim power-performance simulator menghitung secara estimasi daya dari microprocessor dengan menggandakan frekuensi akses ke blok-blok mikroarsitektural fungsional pada model sirkuit dari blok-blok tersebut. Simulator sirkuit elektronis yang populer yakni Falstad, yang dikembangkan oleh Paul Falstad (2015). Terkait keperluan pembelajaran, maka simulator yang disajikan pada artikel ini mengembangkan fungsionalitas dari Falstad, yakni selain kapabilitas simulasi arsitektural yang responsif terhadap fenomena tingkat sirkuit, simulator yang disajikan pada artikel ini juga melengkapi Falstad yakni *offline* dan dukungan terhadap logika CMOS, logika TTL, LED Matrix, dan pemrograman PIC.

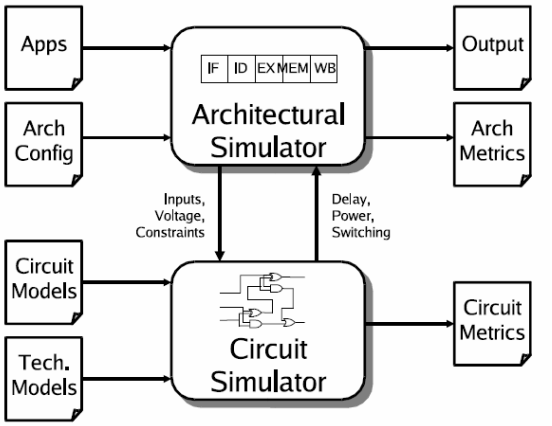
Keunggulan utama dari model-model sirkuit analytical adalah fleksibilitas dan kecepatan. Pada model-model tersebut juga dimungkinkan untuk memiliki berbagai macam konfigurasi hingga tingkat komponen. Idealnya untuk desain arsitektural *circuit-aware*, harus dimiliki akurasi simulasi tingkat sirkuit, dan kecepatan serta fleksibilitas simulasi tingkat arsitektural. Namun desain arsitektural membutuhkan analisis terhadap basis simulasi instruksi dengan jumlah yang sangat banyak, sehingga pada rancang bangun simulator yang disajikan pada artikel ini dilakukan pendekatan perkiraan agresif terhadap perilaku sirkuit guna memenuhi performa yang dibutuhkan.

Faktor Exacta *11 (3):* 275-290, 2018
p-ISSN: 1979-276X
e- ISSN: 2502-339X
DOI : 10.30998/faktorexacta.v11i3.2784

METODE

Simulasi Arsitektural

Arsitektur piranti lunak simulator arsitektural *circuit-aware* ditunjukkan pada gambar 1. Simulator arsitektural ini memiliki dua input yakni model sirkuit terkonfigurasi dan program eksekutor. Arsitektur piranti lunak simulator arsitektural *circuit-aware* ini mengacu pada arsitektur yang dikembangkan oleh T. Austin dan kawan-kawan (2002).



Gambar 1. Arsitektur Piranti Lunak Simulator

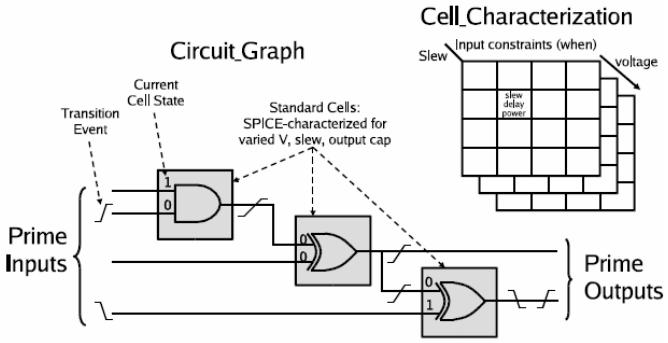
Simulator arsitektural tersebut menghasilkan dua keluaran primer. Jika program mengeksekusi operasi input/output (I/O) apapun (misal akses file, atau input dari *console*), maka operasi-operasi I/O tersebut dieksekusi oleh simulator sebagai bagian dari simulasi. Sebagai tambahan, simulator arsitektural *circuit-aware* pada artikel ini juga memiliki kapabilitas instrumentasi ekstensif yang memungkinkan operasi-operasi saat simulasi terjadi termonitor guna menghasilkan *runtime metrics* seperti *components metrics* (tegangan, arus, daya, dan frekuensi tiap komponen), *bessel vs butterworth output*, *beta saturation and cutoff*, saturasi semikonduktor, tegangan dan arus pada gerbang-gerbang logika, representasi biner sebagai tegangan dalam arsitektur memori, representasi biner sebagai tegangan dalam arsitektur konverter sinyal analog ke sinyal digital dan sebaliknya, *voltage ripple* pada jalur transmisi, parameter-parameter *spark gap*, dan sebagainya. Model simulator arsitektural diimplementasikan dengan SimpleScalar.

Simulasi Sirkuit

Guna mendukung *circuit-awareness* pada simulator arsitektural, maka diimplementasikan simulator sirkuit (dengan bahasa pemrograman Java™) ke dalam model arsitektural. Simulator sirkuit yang diimplementasikan merujuk ke berbagai kombinasi deskripsi logis dari setiap komponen yang disimulasikan, serta terhubung dengan simulator arsitektural pada basis tahap demi tahap. Saat inisialisasi, deskripsi sirkuit dari berbagai komponen dimuat dari kode bahasa pemrograman Java™. Secara bersamaan *interfacing* terhadap model arsitektural yang dikembangkan dengan SimpleScalar dilakukan dengan perangkat penempatan dan *global routing* yang juga dikembangakan dengan bahasa pemrograman Java. Perangkat penempatan dan *global routin*g juga mencakup detail karakteristik *switching* dari blok-blok *cell* standar dari logika yang digunakan pada implementasi fisik model sirkuit dalam simulasi.

Faktor Exacta *11 (3):* 275-290, 2018
p-ISSN: 1979-276X
e- ISSN: 2502-339X
DOI : 10.30998/faktorexacta.v11i3.2784

Pada setiap daur simulasi, setiap blok logika diberikan *vector* masukan baru dari simulator arsitektural. Setiap *vector* terkait dengan himpunan nilai milik *pipeline* dari setiap tahap masukan. Dengan demikian simulator sirkuit dapat melakukan komputasi pengukuran-pengukuran yang relevan sesuai kebutuhan analisis terhadap proses simulasi spesifik. Tergantung dari tujuan simulasi, pengukuran-pengukuran yang dilakukan saat proses simulasi akan dikembalikan ke simulator arsitektural guna mengarahkan kemajuan dari simulasi atau kembali ke simulator arsitektural sebagai keluaran untuk dievaluasi. Simulator sirkuit memiliki akurasi yang cukup sebagai perangkat analisis yang mampu menguji *transient fault injection* serta melakukan investigasi terhadap variasi-variasi proses. Gambaran metodologi simulasi sirkuit ditunjukkan pada gambar 2.



Gambar 2. Metodologi Simulasi Sirkuit

Implementasi simulator sirkuit adalah berbasis pada skema *event-driven*. Setiap aktivitas di dalam logika sirkuit direpresentasikan sebagai transisi yang terdiri atas nilai logika akhir, waktu kedatangan, dan *slew* (*slope* transisi tegangan). Deskripsi skematis dari model sirkuit yang digunakan oleh simulator ditunjukkan pada gambar 2. Pada setiap daur simulasi, himpunan pertama dari transisi dihasilkan dari perbandingan himpunan masukan baru dengan himpunan masukan dari daur sebelumnya. Dalam setiap himpunan, semua transisi masukan memiliki waktu kedatangan yang sama dan besaran *slew* yang tetap. Jika terjadi transisi pada sebuah *cell* dari jalur masukan, fungsi dari *cell* dievaluasi dan sebuah transisi baru dapat atau tidak dapat dihasilkan pada keluaran *cell* (tergantung fungsi logika *cell*). Transisi baru yang dihasilkan ditambahkan pada antrian *event* terurut secara *ascending* berdasar waktu kedatangan. Berdasarkan percobaan terhadap simulator yang tersaji pada artikel ini, ditemukan hanya antara 2% hingga 10% dari *cell* sirkuit harus dievaluasi, sehingga pendekatan simulasi yang diterapkan dapat dikatakan sebanding dengan simulator-simulator logika komersial. Guna memenuhi performa yang dipersyaratkan pada simulator arsitektural, maka tegangan direpresentasikan dengan nilai logika biner yakni 0 dan 1. Antrian *event* terurut memungkinkan pencegahan kesalahan pendekatan dengan eliminasi pasangan transisi berlawanan yang terjadi di dalam sebuah interval yang lebih singkat dari waktu transisi penuh. *Pseudo-code* dari algoritma simulasi ditunjukkan pada gambar 3.

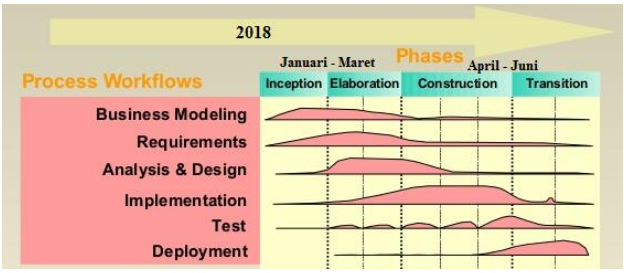
```
Circuit::Simulate( vector Input )  {
    power = 0;
    for each (changedBit in Input) {
        Tr = GenInputTransition(changedBit);
        EventQueue.InsertSorted(Tr);
    }
    while (!EventQueue.empty()) {
        Tr = EventQueue.RemoveEarliest();
        for each (cell, Tr.net ∈ cell.Inputs) {
            power += cell.Evaluate(Tr);
        }
    }
}
```

Gambar 3. Algoritma Simulasi Sirkuit

Pada saat konfigurasi, dilakukan komputasi tabel karakterisasi untuk tiap *cell*. Tabel tersebut yang ditunjukkan pada gambar 2 mendukung komputasi transisi keluaran saat terjadi simulasi. Setiap tabel karakterisasi memiliki parameter berdasar tegangan masukan (V_{dd}) dari sistem, *slew* transisi masukan, dan batasan transisi masukan (0 ke 1 atau 1 ke 0). Tabel karakterisasi menyediakan *slew* dan *delay* transisi pada keluaran, serta konsumsi daya untuk setiap *cell* terkait *event*. Guna menghasilkan tabel karakterisasi, terlebih dahulu dilakukan komputasi kapasitas dari setiap cell. Kapasitas tiap *cell* diperoleh dengan mengkombinasikan dua komponen kapasitif sesuai panjang kawat, yang mana kesemuanya tersedia oleh mekanisme *routing* yang diimplementasikan pada perangkat lunak simulator sirkuit yang dikembangkan, berikut mekanisme *fanout* untuk setiap *cell*. Kapasitas *fanout* diperoleh dengan menambahkan kapasitas pada setiap elemen *fanout* dari *cell*. Setelah kapasitas keluaran *cell* terkomputasi, maka tabel karakterisasi dari setiap *cell* dapat diperoleh. Tabel karakterisasi dari setiap *cell* menyediakan *delay* dan data daya berdasar kapasitas, tegangan, dan tipe transisi. Kemudian tabel karakterisasi digunakan pada saat simulasi guna komputasi ukuran-ukuran yang relevan untuk setiap transisi.

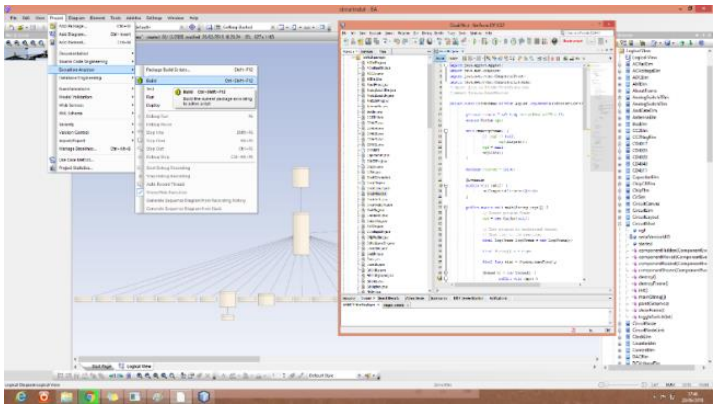
Rekayasa Piranti Lunak

Piranti lunak simulator arsitektural *circuit-aware* pada artikel ini dikembangkan dengan framework proses pengembangan piranti lunak Rational Unified Process (RUP), yang dirumuskan oleh Sten Jacobson (2002). Chart yang mengilustrasikan alur implementasi RUP ditunjukkan pada gambar 4. RUP diimplementasikan agar supaya fase-fase pada proses rancang bangun dapat terlaksana secara prediktif; serta dikarenakan penelitian ini dilaksanakan dengan dana yang terbatas maka RUP yang menyediakan rencana spesifik untuk setiap fase memungkinkan penghematan sumber daya sekaligus mengurangi biaya - biaya rancang bangun yang tidak diharapkan. Selain mengimplementasikan RUP, proses rancang bangun piranti lunak simulator arsitektural *circuit-aware* juga menerapkan paradigma pemrograman berorientasi objek dengan bahasa pemrograman Java™. Alat bantu pemodelan pada fase analisis dan perancangan yakni Unified Modelling Language (UML). Pemodelan yang diterapkan adalah pemodelan berorientasi objek dari detail sistematika piranti lunak berikut pemodelan data. Pemodelan yang diterapkan menggunakan sudut pandang logika dengan mengimplementasikan diagram kelas (*class diagram*), yang mana diagram kelas tersebut tidak dapat ditampilkan pada artikel ini dikarenakan ukurannya yang sangat besar. Berdasarkan pemodelan dengan diagram kelas maka piranti lunak simulator arsitektural *circuit-aware* pada artikel ini terdiri atas 122 *class* dan 1 *editable interface class*.



Gambar 4. Ilustrasi Alur Implementasi RUP

Setelah dilakukan pemodelan dari sudut pandang logika menggunakan diagram kelas, selanjutnya dilakukan *forward engineering* terhadap model sistematika detail tersebut menjadi kode pemrograman (*skeleton code*) dalam bahasa pemrograman Java™. Pemodelan dan *forward engineering* menggunakan perangkat Enterprise Architect, sedangkan pengembangan piranti lunak/pemrograman dari *skeleton code* menggunakan Integrated Development Environment (IDE) Netbeans. Proses *forward engineering* ditunjukkan pada gambar 5.

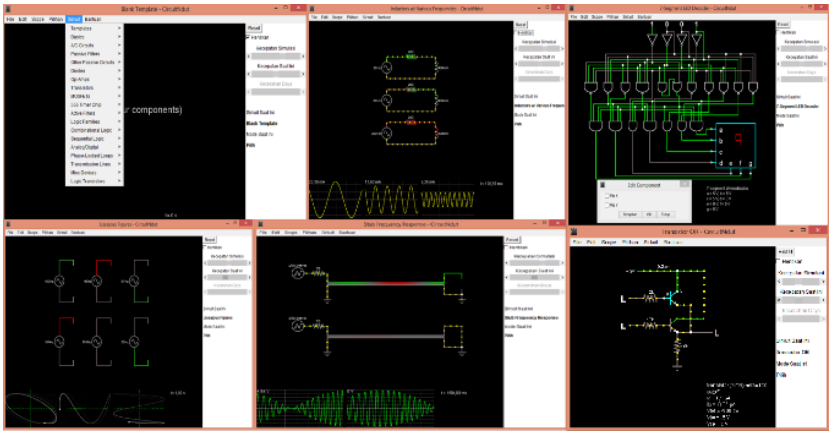


Gambar 5. Proses *Forward Engineering*

Hasil Pengembangan

Piranti lunak simulator arsitektural *circuit-aware* pada artikel ini dikembangkan dengan tujuan utama sebagai alat bantu pembelajaran, oleh karena itu guna mempermudah pengguna maka pada simulator arsitektural *circuit-aware* ini terdapat berbagai *template* sirkuit elektronis yang telah dibangun sebelumnya. Pengguna dapat mengubah template sirkuit yang telah ada maupun membuat sirkuit baru dari kondisi kosong. Semua simulasi sirkuit yang dilakukan pada simulator secara otomatis menampilkan analisis pada level arsitektural. Contoh fungsionalitas piranti lunak simulator arsitektural *circuit-aware* yang disajikan pada artikel ini ditunjukkan pada gambar 6. Simulator arsitektural *circuit-aware* ini mendukung berbagai macam tipe sirkuit baik yang terklasifikasi dalam sirkuit arus kuat, serta arus lemah baik analog maupun digital.

Faktor Exacta *11 (3):* 275-290, 2018
p-ISSN: 1979-276X
e- ISSN: 2502-339X
DOI : 10.30998/faktorexacta.v11i3.2784



Gambar 6. Contoh Fungsionalitas Piranti Lunak Simulator Arsitektural *Circuit-Aware*

Optimasi Performa

Constraint-Based Circuit Pruning

Seringkali pada analisis arsitektural, membutuhkan berjalannya simulasi yang hanya melakukan pengukuran terhadap satu hal yang relevan pada simulator sirkuit, dimana hal terukur tersebut dianggap relevan jika memiliki nilai diatas batasan yang ditetapkan sebelumnya. Hal ini terjadi pada teknik-teknik *throttling* dan Razor *clocking*. Sebagai contoh untuk arsitektur dengan Razor *clocking*, simulasi sirkuit harus dievaluasi jika *delay* propagasi diatas atau dibawah nilai waktu daur *clock*, sedangkan nilai aktual dari delay propagasi sendiri tidak relevan. Pada umumnya analisis *static* pada *domain-specific* dapat diterapkan untuk komputasi kemungkinan terburuk dari batasan nilai ukuran, dimana kemudian dilakukan *pruning* terhadap simulasi guna eliminasi porsi *netlist* dari sirkuit dimana batasan tidak boleh dilanggar. Secara spesifik pada desain Razor, semua jalur jaringan logika yang memiliki *delay* propagasi dibawah batasan *clock* yang ditetapkan pasti dapat dihapus karena jalur tersebut tidak mempengaruhi simulasi arsitektural. Dengan menerapkan pendekatan ini, maka performa simulator arsitektural *circuit-aware* pada artikel dapat ditingkatkan tanpa mempengaruhi akurasi.

Guna proses komputasi *cell* mana yang dapat dihilangkan dari jaringan logika, terlebih dahulu dikomputasikan kemungkinan terburuk untuk *delay* dari masukan ke setiap keluaran. *Delay* propagasi melalui *cell* ditemukan pada setiap basis tegangan dari tabel karakterisasi. Setelah kemungkinan terburuk dari *delay* pada setiap keluaran diketahui, maka kemudian dapat dihapus dari netlist sehingga menjamin stabilitas spesifikasi waktu daur pada keluaran berikut *cell-cell* spesifik yang memiliki kontribusi terhadap keluaran. Setiap kali terjadi perubahan tegangan pada simulasi sirkuit, dihitung kembali *constraint-based pruning* sebelum melanjutkan simulasi. Penerapan pruning juga didukung oleh optimasi lain yakni menahan parameter pada setiap *node* internal dari *netlist* pada nilai maksimum. Saat simulasi mencapai *node* tersebut, jika parameter yang dievaluasi memiliki nilai yang lebih rendah dari batasan yang diperbolehkan maka secara otomatis simulasi dipaksa mencapai *net-net* dari keluaran tanpa melanggar batasan. Dengan demikian simulasi dapat berlanjut dari node internal tersebut ke *switching* keluaran dalam bentuk simulasi logika murni tanpa proses komputasi untuk simulasi ukuran-ukuran yang lain. Peningkatan performa dengan *pruning* tergantung dari seberapa ketat batasan yang ditetapkan (misalnya level tegangan). Pada simulator arsitektural *circuit-aware* yang disajikan pada artikel ini, *pruning* dapat meningkatkan kecepatan simulasi hingga hampir dua kali lipat walau simulasi dilakukan pada level tegangan rendah (1,4V).

Circuit Timing Memorization

Lokalitas merupakan kunci utama dalam perancangan *microarchitectures* modern. Lokalitas secara prinsip yakni instruksi-instruksi program dan data yang baru-baru saja digunakan, memiliki kemungkinan yang lebih besar untuk digunakan kembali di masa depan dibanding nilai acak. Oleh karena itu seperti dinyatakan oleh M.H. Lipasti dan G. S. Ravi (2017), perangkat-perangkat seperi *cache* dan *value predictor* dapat secara akuran mengantisipasi kebutuhan program dari program berdasar aktivitas yang pernah dilakukan. Lokalitas dapat dimanfaatkan guna meningkatkan performa simulasi pewaktuan sirkuit (*circuit timing*). Implementasinya adalah dengan membangun tabel *hash* yang merekam (*memorizes*) formula pemetaan untuk setiap modul level sirkuit. Formula pemetaan tersebut yakni:

$$(\mathbf{vector}_{state}, \mathbf{vector}_{in}, V_{dd}) \rightarrow (\mathbf{delay}, \mathbf{energy})$$

dimana $\mathbf{vector}_{state}$ merupakan kondisi sirkuit saat ini, $\mathbf{vector}_{in}$ merupakan vector masukan saat ini, dan V_{dd} adalah tegangan operasi saat ini. Tabel *hash* menghasilkan evaluasi *latency* sirkuit dan evaluasi energi sirkuit. Tabel *hash* terindeks dengan kombinasi $\mathbf{vector}_{state}$ dan $\mathbf{vector}_{in}$. $\mathbf{Vector}_{state}$ *encode* keadaan sirkuit saat ini dan $\mathbf{vector}_{in}$ mengindikasikan transisi masukan. Masukan dari tabel *hash* dikombinasikan dengan tegangan operasi saat ini (V_{dd}) *encodes* faktor-faktor yang menentukan *delay* dan energi secara keseluruhan. Kapanpun jika tabel *hash* tidak mencakup *entry* yang diminta, maka kemudian dilakukan simulasi sirkuit skala penuh guna komputasi *delay* dan energi dari komputasi sirkuit. Hasil dari simulasi skala penuh tersebut selanjutnya dimasukkan ke dalam tabel *hash* dengan harapan bahwa untuk selanjutnya aplikasi simulator akan menghasilkan *vector-vector* serupa. Ukuran tabel *hash* yang diimplementasikan dibatasi sebesar 256 MB.

Performa simulasi yang lebih baik dicapai dengan *re-order* rantai *hash bucket* secara dinamis, yakni dengan meletakkan elemen yang dirujuk terkini pada ujung awal rantai, serta *exploit* nilai lokalitas pada program.

Dengan implementasi *baseline* dari tabel *hash*, maka diperoleh tipikal *hit rate* kurang dari 50%, yang mana berarti kecepatan simulasi masih memungkinkan untuk ditingkatkan. Walau demikian *hashing* terhadap keseluruhan *vector* masukan ke tahap logika *pipeline* akan menimbulkan *overhead*. Sebagai contoh instruksi-instruksi *load* yang melalui tahap eksekusi (EX) *pipeline* mencakup dua *register operand* pada *vector* masukannya. Saat *load* dieksekusi, *operand* ke dua diabaikan karena digunakannya *offset field* dari instruksi. Dengan mencakup *operand* ke dua dari *vector* masukan, maka dibutuhkan jumlah entry dari tabel hash yang berlipat ganda guna perekaman alamat komputasi yang sama. *Overhead* tersebut diatasi dengan mekanisme penyaringan *vector* masukan per-*opcode*, dimana setiap *opcode* dari instruksi diberi *mask* dimana dinyatakan bahwa masukan tidak mempengaruhi evaluasi tahap logika. Masukan diberi *mask* sebelum perekaman simulasi sirkuit. Penyaringan dengan *mask* ini meningkatkan rerata *hit rate* pada tabel *hash* sebesar 70-85% sehingga meningkatkan kecepatan simulasi antara tiga hingga lima kali bergantung dari apa yang disimulasikan

Analisis SimPoint

Setelah mekanisme-mekanisme optimasi diterapkan, maka simulasi dapat dijalankan dengan kecepatan 1000 instruksi per detik. Visualisasi performa simulasi untuk simulator arsitektural *circuit-aware* yang disajikan pada artikel ini menggunakan analisis SimPoint pada

perangkat lunak SimPoint. SimPoint digunakan dikarenakan dapat mensimulasikan performa program pada *microarchitecture* berdasar penelitian oleh T. Sherwood dan kawan-kawan (2013). SimPoint menggunakan analisis distribusi blok berikut teknik lain seperti analisis *clustering* hingga penyimpulan secara tepat terhadap perilaku dari bagian-bagian eksekusi program. Analisis performa sekaligus visualisasi dengan SimPoint mengurangi waktu simulasi secara signifikan dengan hanya menggunakan *sample-sample* yang mewakili.

Terkait analisis performa simulator arsitektural *circuit-aware* yang disajikan pada artikel ini, digunakan *sample* yang dikembangkan pada penelitian oleh T. Sherwood dan kawan-kawan (2017), dengan panjang 10 juta instruksi atau yang kemudian menjadi dukungan Early Multiple SimPoints dari perangkat lunak SimPoint. SimPoint melakukan analisis dengan mengkombinasikan titik awal *sample* simulasi program, panjang *sample*, serta instruksi per *clock/cycle*. B. Calder (2003) menyatakan bahwan analisis kesalahan dengan SimPoint menyatakan rerata kesalahan untuk berbagai macam *benchmark* yang dapat dilakukan adalah kurang dari 10%.

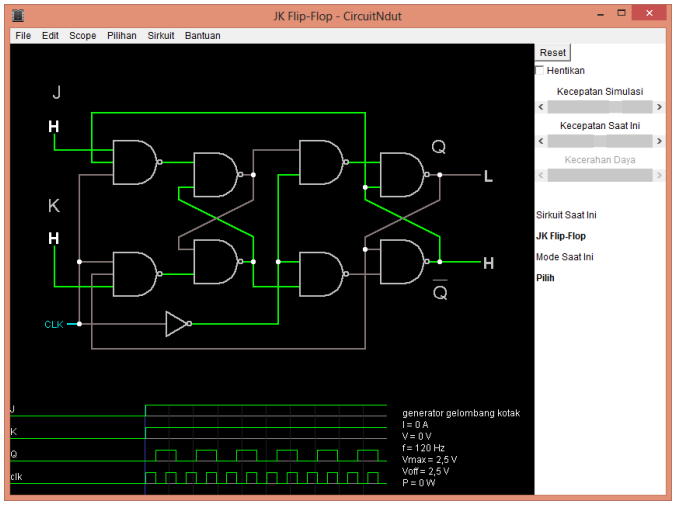
Studi Kasus

Guna evaluasi kualitas simulator arsitektural *circuit-aware*, maka dimodelkan teknik Razor clocking dari penelitian oleh D. Ernst dan kawan-kawan (2003), dimana *latency* dari sebuah instruksi (dalam *cycles*) dapat bervariasi tergantung dari *latency* evaluasi sirkuit di dalam tahapan *pipeline*. Pada studi kasus ini dilakukan evaluasi terhadap teknologi Razor clocking dengan simulator arsitektural *circuit-aware* yang telah dikembangkan.

Spekulasi Pewaktuan Razor

Kunci dari observasi terhadap desain Razor adalah tidak dimungkinkannya skenario terburuk dikarenakan kemampuan mekanisme deteksi dan perbaikan kesalahan pada desain Razor, bahkan hingga *tuning* kebutuhan energi tipikal. Desain yang dihasilkan pada Razor memiliki kebutuhan energi yang lebih rendah secara signifikan bahkan saat perbaikan kesalahan. Desain Razor memiliki pewaktuan deteksi kesalahan *in-situ* serta mekanisme koreksi kesalahan terimplementasi dalam Razor *flip-flop*. Razor *flip-flop* meningkatkan kehandalan dengan menggandakan nilai besaran *clock* dari *clock* cepat menjadi *clock* tertunda. Dengan kondisi *clock* tertunda maka sirkuit deteksi kesalahan *metastability-tolerant* dapat difungsikan guna memeriksa validitas semua nilai pada tahap *pipeline*. Jika terjadi kesalahan pewaktuan maka mekanisme *pipeline flush* termodifikasi akan mengembalikan nilai yang benar ke dalam *pipeline*, menghapus instruksi-instruksi sebelumnya, dan memulai kembali instruksi berikutnya setelah komputasi yang keliru dibatalkan. Desain simulasi Razor *flip-flop* pada simulator arsitektural *circuit-aware* yang dikembangkan pada penelitian ini ditunjukkan pada gambar 7.

Faktor Exacta *11 (3):* 275-290, 2018
p-ISSN: 1979-276X
e- ISSN: 2502-339X
DOI : 10.30998/faktorexacta.v11i3.2784



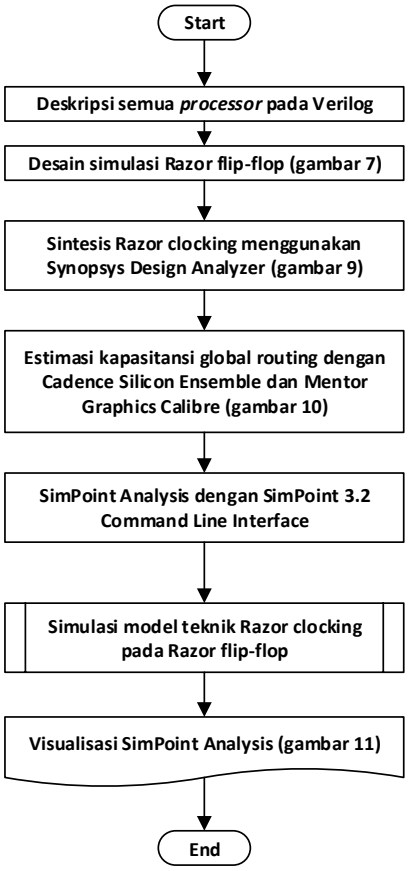
Gambar 7. Desain Simulasi Razor *flip-flop*

Lingkungan Percobaan

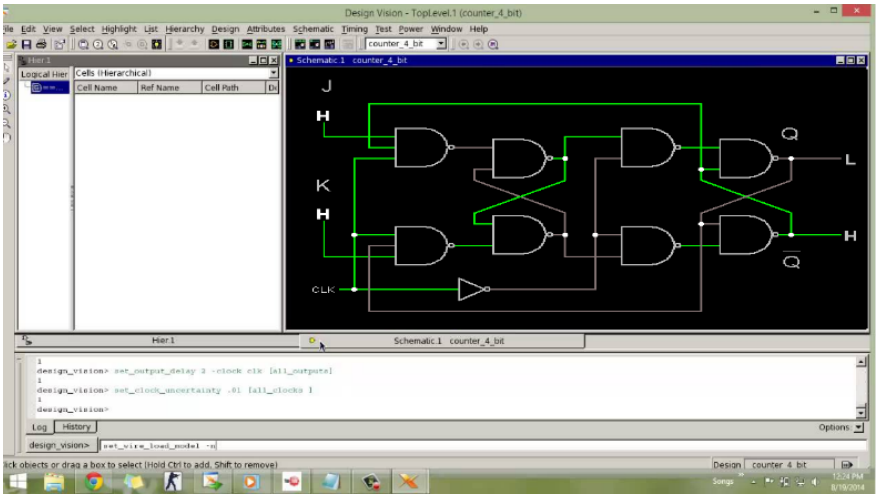
Guna memodelkan Razor clocking, diimplementasikan model arsitektural dari sebuah *baseline* model *processor* Alpha 64-bit. Arsitektur dari *processor* Alpha tersebut sederhana, dimana memiliki *pipeline* yang terdiri atas *instruction fetch*, *instruction decode*, *execute*, dan *memory/writeback* dengan I-cache dan D-cache sebesar 8 Kbytes. Keseluruhan processor terdeskripsi dalam Verilog dan tersintesis menggunakan Synopsys Design Analyzer (versi 2003.03-2). Kapasitansi *global routing* diestimasikan dengan melakukan penempatan global dan *routing* menggunakan Cadence Silicon Ensemble (versi 5.4.126) dan Mentor Graphics Calibre (versi 9.1_5.6). Processor dipetakan ke sebuah proses Taiwan Semiconductor Manufacturing Company (TSMC) 0.18um dan tervalidasi untuk beroperasi pada 200 MHz. Setelah dilakukan analisis performa, diketahui bahwa hanya tahap-tahap *instruction decode* dan *execute* yang terpengaruh *setting* tegangan dan frekuensi, sehingga hanya tahap-tahap tersebut yang dicakup pada simulasi arsitektural *circuit-aware*.

SimPoint dijalankan untuk komputasi hingga 30 titik simulasi menggunakan pencarian biner untuk sebuah inisialisasi *seed* tunggal pada setiap clustering sehingga pada proses percobaan pemodelan teknik Razor clocking, SimPoint dijalankan melalui Command Line Interface (CLI) dengan perintah `simpoint -maxK 30 -numInitSeeds 1 -loadFVFile gcc-00-166-ref.` Opsi `-loadFVFile gcc-00-166-ref` artinya SimPoint memuat file simulasi yang menyatakan Razor *flip-flop* berjalan pada processor yang melakukan komputasi GCC (Gnu C Compiler), dimana GCC tersebut menjalankan program yang dinamakan 166-ref.c. Proses percobaan dengan pemodelan teknik Razor clocking dipaparkan pada *flowchart* di gambar 8.

Faktor Exacta *11 (3):* 275-290, 2018
p-ISSN: 1979-276X
e- ISSN: 2502-339X
DOI : 10.30998/faktorexacta.v11i3.2784

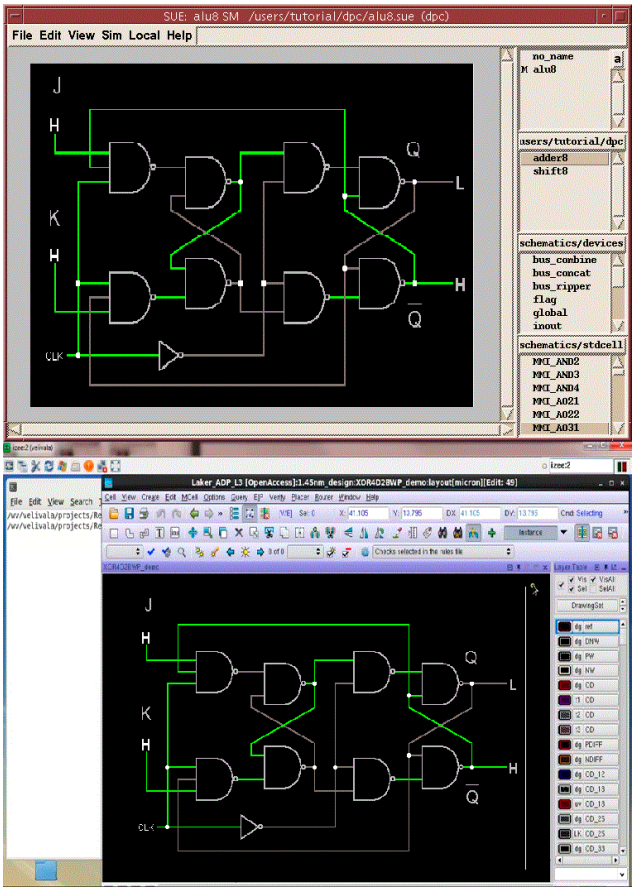


Gambar 8. *Flowchar*t Proses Percobaan



Gambar 9. Sintesis *Razor Clocking*

Faktor Exacta *11 (3):* 275-290, 2018
p-ISSN: 1979-276X
e- ISSN: 2502-339X
DOI : 10.30998/faktorexacta.v11i3.2784



Gambar 10. Estimasi Kapasitansi *Global Routing*

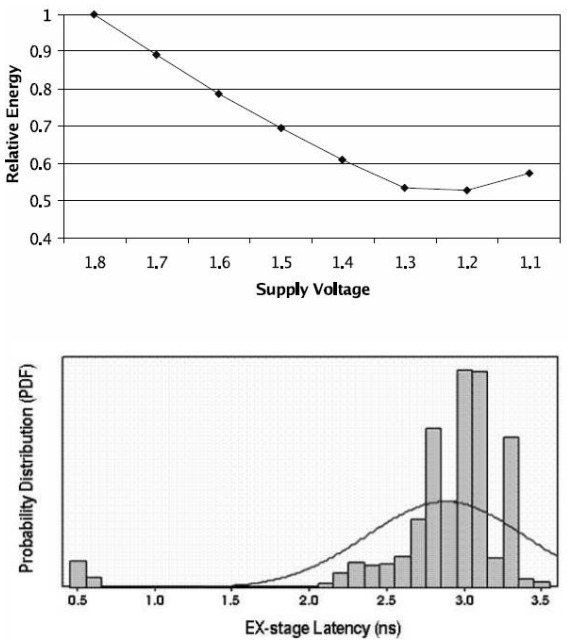
Studi Kasus Simulasi

Tabel 1 menunjukkan *baseline* performa dari piranti lunak simulator arsitektural *circuit-aware* yang mensimulasikan desain razor pada 50 instruksi per detik. Namun setelah optimasi diterapkan, simulator arsitektural *circuit-aware* ini dapat mencapai 887 instruksi per detik, atau sekitar delapan kali lebih cepat dibandingkan Verilog.

Tabel 1. Manfaat Optimasi Pada Simulasi Sirkuit (studi kasus: simulasi GCC)

Opsi Optimasi	**nstruksi per Detik**
tidak ada	102
prunning	347
running dan memoizati*on*	887

Faktor Exacta *11 (3):* 275-290, 2018
p-ISSN: 1979-276X
e- ISSN: 2502-339X
DOI : 10.30998/faktorexacta.v11i3.2784



Gambar 11. Visualisasi *SimPoint Analysis* terhadap Studi Kasus Desain Razor

Gambar 11 menunjukkan performa *Razor clocking* yang diukur dengan piranti lunak simulator arsitektural *circuit-aware* yang dikembangkan pada penelitian ini. Grafik atas menunjukkan energi relatif dari *pipeline* dengan tegangan yang menurun. Seiring dengan penurunan tegangan, energi *pipeline* tersimulasi juga menurun, walau dengan adanya *recovery* kesalahan pewaktuan. Karena simulator arsitektural *circuit-aware* yang dikembangkan pada penelitian ini dapat secara akurat mengukur tahap evaluasi *latency* per-*cycle*, sehingga dimungkinkan untuk mengakses tegangan dimana beban *recovery* kesalahan pewaktuan dari Razor lebih besar dibanding manfaat penurunan tegangan (sekitar 1,2V). Sebagai tambahan grafik bawah pada gambar 11 menunjukkan kemampuan pengukuran dari simulator arsitektural *circuit-aware* yang dikembangkan pada penelitian ini, dimana menunjukkan *latency* melalui tahap logika EX, sebagai sebuah fungsi distribusi probabilitas untuk semua vektor masukan selama simulasi GCC. Kemungkinan terburuk dari *latency* pada tahap evaluasi selama simulasi GCC adalah 4ns, sehingga terlihat pada gambar 11 bahwa tipikal kasus *latency* adalah jauh dibawah 4ns sehingga memungkinkan desain Razor pada simulator arsitektural *circuit-aware* yang dikembangkan pada penelitian ini untuk menurunkan tegangan dengan hanya sedikit menambah laju kesalahan pewaktuan sirkuit.

PENUTUP

Simpulan

Simulator sirkuit elektronis dengan dukungan simulasi arsitektural yang dikembangkan pada penelitian ini mampu mensimulasikan berbagai macam sirkuit baik analog maupun digital, serta baik pada ranah arus kuat maupun arus lemah; dimana pada kesemua sirkuit yang disimulasikan dapat dilakukan pengukuran hingga *level* arsitektural. Fitur *template* sirkuit dan contoh sirkuit-sirkuit terbangun yang bertujuan untuk mempermudah pembelajaran juga berhasil diimplementasikan. Artikel ini menunjukkan bahwa dimungkinkan untuk mengkombinasikan simulasi sirkuit dengan simulator arsitektural, dimana tetap dapat mencapai laju keluaran

simulasi yang signifikan. Dengan identifikasi *event-event* berulang atau non-kritis, *delay* terhadap informasi-informasi yang bersifat *voltage dependent* dan *data dependent* akan dapat terbaca; yang mana biasanya analisis semacam ini membutuhkan coupling yang rumit dari simulasi arsitektural dan simulasi SPICE (Simulation Program With Integrated Circuit Emphasis). Artikel ini merumuskan solusi otomatis terhadap masalah spesifik terkait *delay* terhadap *voltage dependent* dan *data dependent*; yang dapat dikembangkan untuk *run-time dependencies* yang lain seperti variasi proses dan *coupling* derau.

Ucapan Terima Kasih

Penelitian yang sedang berjalan ini dibiayai oleh Direktorat Riset dan Pengabdian Masyarakat Direktorat Jenderal Penguatan Riset dan Pengembangan Kementerian Riset, Teknologi, dan Pendidikan Tinggi Nomor: 044/KM/PNT/2018, Tanggal 6 Maret 2018.

DAFTAR PUSTAKA

Austin, T., Larson, E., dan Ernst, D., Februrari 2002. *Simplescalar: An infrastructure for computer system modeling*. IEEE Computer.

Cai, G., Seng, John S., dan Tullsen, Dean M., 30 September – 3 Oktober 2012. *Retrospective on "Power-Sensitive Multi Threaded Architecture"*. In 2012 IEEE 30th International Conference on Computer Design (ICCD).

Calder, B., 2003. Simpoint, [online] Tersedia di: <http://www.cse.ucsd.edu/calder/simpoint/> [Diakses 28 Juni 2018]

Ernst, D., Kim, N.S., Das, S., Pant, S., Pham, T., Rao, R., Ziesler, C., Blaauw, D., Austin, T., Mudge, T., dan Flautner, K., Desember 2003. *Razor: A low-power pipeline based on circuit-level timing speculation*. 36th Annual International Symposium on Microarchitecture (MICRO-36).

Falstad, Paul., 2015. Circuit Simulator Applet, [online] Tersedia di: <http://www.falstad.com/circuit/> [Diakses 9 Juni 2018]

Ghiasi, S. dan Grunwald, D., Desember 2000. *A comparison of two architectural power models. Workshop on Power Aware Computing Systems* (PACS-2000).

Jacobson, Sten., 2002. The Rational Objectory Process–A UML Based Software Engineering Process, [online] Tersedia di: <http://www.iscn.at/select_newspaper/object/rational.html> [Diakses 18 Juni 2018]

Lipasti, M.H. dan Ravi, G.S., 24 – 28 Juni 2017. *CHARSTAR: Clock hierarchy aware resource scaling in tiled architecture*. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA).

Sherwood, T., McMahan, J., Cui, W., Xia, L., Heckey, J., dan Chong, Frederic T., 1 – Mei 2017. *Challenging on-chip SRAM security with boot-state statistics*. 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST).

Sherwood, T., Kastner, R., dan Oberg, J., 2013. *Eliminating Timing Information Flows in a Mix-Trusted System-on-Chip*. IEEE Design & Test – Volume: 30, Issue: 2 – 2013 – pages: 55-62.

Kevin Skadron, Alec Roelke, Runjie Zhang, Kaushik Mazumdar, Ke Wang, Mircea R. Stan, 5 – 8 November 2017. *Pre-RTL Voltage and Power Optimization for Low Cost, Thermally Challenged Multicore Chips*. 2017 IEEE International Conference on Computer Design (ICCD).

Weaver, C. dan Austin, T., Juni 2001. *A fault tolerant approach to microprocessor design*. IEEE International Conference on Dependable Systems and Networks (DSN-2001).