# PERANCANGAN *DIVIDER* 8-BIT DENGAN TEKNOLOGI 180NM MENGGUNAKAN PERANGKAT LUNAK ELECTRIC

Rizko Prasada Fitriansyah[*)], Munawar Agus Riyadi, and Muhammad Arfan

Departemen Teknik Elektro, Universitas Diponegoro, Semarang
Jl. Prof. Sudharto, SH, Kampus UNDIP Tembalang, Semarang 50275, Indonesia

[*)]*E-mail: rizko.prasada.F@gmail.com*

## Abstrak

Operasi pembagian merupakan salah satu operasi penting yang ada pada blok *Arithmetic Logic Unit* (ALU). Meskipun operasi pembagian lebih jarang digunakan jika dibandingkan dengan penjumlahan dan perkalian, lamanya waktu yang dibutuhkan untuk menyelesaikan operasi ini menyebabkan banyak energi yang terbuang. Untuk mengatasi permasalahan tersebut, terdapat beberapa teknik yang dapat dilakukan, salah satunya adalah dengan memilih algoritma yang tepat sehingga operasi yang dilakukan juga semakin cepat. Pada penelitian ini, dirancang sebuah *divider* menggunakan teknologi 180nm dengan algoritma *non-restoring*. Dalam penerapannya, digunakan perangkat lunak Electric untuk merancang *layout* dan LTspice untuk menguji fungsional serta melakukan pengukuran *timing delay*. Dari perancangan yang dilakukan, didapati *divider* ini memiliki luas sebesar 0,027mm$^2$, *propagation delay time* sebesar 3,644ns, dan *area coverage* sebesar 45,975%.

*Kata kunci: divider, 180nm, algoritma non-restoring*

## Abstract

Division operation is one of important operations in Arithmetic Logic Unit (ALU) block. Although division operation is an infrequent operation compared to addition and multiplication, its longer latency makes division dissipate a significant amount of energy. There are some techniques to solve the problem, one of them is a precise algorithm choosing so the operation will be faster. In this research, a divider using 180nm technology based on non-restoring algorithm is designed. In application, this research used Electric to design the layout and LTspice to do functional test and to measure timing delay. As a result, this divider has 0,0027mm$^2$ for area, 3,644ns for *propagation delay time*, and 45,975% for area coverage.

*Keywords: divider, 180nm, non-restoring algorithm*

## 1. Pendahuluan

Perancangan rangkaian terpadu (*integrated circuit*, disingkat IC) memiliki peran penting dalam menjawab tuntutan zaman yang menginginkan teknologi serba lebih baik. Peningkatan kecepatan akses, pengurangan konsumsi daya, dan pengecilan ukuran adalah beberapa upaya yang dapat dilakukan untuk memperoleh teknologi terbaik. Upaya-upaya tersebut berkaitan erat dengan perancangan rangkaian terpadu, lebih lanjut disebut teknologi IC khususnya *Very-large-scale integration* (VLSI).

Salah satu penerapan VLSI adalah pada prosesor. Untuk dapat melakukan pemrosesan data, prosesor terdiri dari beberapa blok utama. Salah satu blok terpenting dari prosesor di antaranya adalah *Arithmetic Logic Unit* (ALU). ALU merupakan blok yang berfungsi untuk melakukan operasi perhitungan seperti penjumlahan, pengurangan, pengalian, pembagian, penggeseran bit, dan lain-lain.

Operasi pembagian merupakan salah satu operasi dasar yang ada pada blok ALU selain operasi penjumlahan, pengurangan, dan perkalian. Meskipun operasi pembagian lebih jarang digunakan jika dibandingkan dengan penjumlahan dan perkalian, lamanya waktu yang dibutuhkan untuk menyelesaikan operasi ini membuat banyak energi yang terbuang [1]. Untuk mengatasi permasalahan tersebut, terdapat beberapa teknik yang dapat dilakukan, salah satunya adalah dengan memilih algoritma yang tepat sehingga operasi yang dilakukan juga semakin cepat.

Terdapat banyak algoritma yang dapat digunakan untuk melakukan proses pembagian. Penelitian yang dilakukan

oleh Siba Kumar Panda dan Arati Sahu dengan judul "*A Novel Vedic Divider Architecture with Reduced Delay for VLSI Applications*" menggunakan metode *Vedic mathematic-Parvatya Sutra*. Penelitian ini disimulasikan menggunakan Xilinx ISE Simulator dan diimplementasikan pada virtex4 FPGA divais XC4VLX15 [2]. Penelitian lain dilakukan oleh C.-L.Wey dan C.-P.Wang dengan judul "*Design of a Fast Radix-4 SRT Divider and its VLSI Implementation*" yang menggunakan metode estimasi [3]. Selain itu, terdapat pula perancangan *divider* dengan menggunakan algoritma *non-restoring* oleh Pradeep Nair, Dhireesha Kudithipudi *et al*, namun tidak dijelaskan secara rinci seperti apa kinerja *divider* tersebut [1].

Penelitian ini bertujuan untuk merancang sebuah *divider* menggunakan teknologi 180nm dengan algoritma *non-restoring* melalui perangkat lunak Electric. Rancangan tersebut kemudian diuji secara fungsional dan diukur seberapa besar luas, *delay*, dan *area coverage*-nya.

## 2. Metode

### 2.1. Pembagian Bilangan Biner Algoritma *Non-restoring*

Terdapat beberapa metode atau algoritma yang dapat digunakan untuk melakukan operasi pembagian pada bilangan biner. Secara garis besar, algoritma tersebut dikelompokkan menjadi dua berdasarkan operasi perulangannya (*iterative operator*). Algoritma pertama menggunakan operasi perulangan pengurangan, sementara algoritma kedua menggunakan operasi perulangan perkalian [4]. Algoritma *non-restoring* termasuk ke dalam algoritma jenis pertama.

Seperti telah disebutkan sebelumnya, terdapat istilah operasi perulangan. Maksud operasi perulangan di sini adalah, operasi tersebut (dalam algoritma *non-restoring* berarti pengurangan) dilakukan berulang-ulang hingga mencapai hasil akhir. Untuk lebih jelasnya, berikut merupakan algoritma *non-restoring* yang digunakan untuk perancangan *divider* ini [5].

1. Mulai
2. Tentukan nilai pembagi (*divisor*) dan yang dibagi (*dividend*)
3. Ubah *divisor* menjadi bilangan *2's complement*-nya
4. Susun nilai *partial remainder* awal, yaitu bilangan 0 sebanyak *n*-bit diikuti dengan bilangan *dividend*-nya (*n* merupakan besarnya bit pada *divisor* dan *dividend*)
5. Geser nilai *partial remainder* ke kiri sebesar 1 bit
6. Jumlahkan nilai *partial remainder* yang sudah digeser ke kiri sebesar 1 bit (langkah 5) dengan *divisor* yang sudah diubah menjadi bilangan *2's complement*-nya (langkah 2)
7. Jika hasil penjumlahan tidak menghasilkan *carry out*, maka nilai *partial remainder* yang digunakan untuk penjumlahan selanjutnya adalah nilai *partial remainder* yang digunakan sebelumnya dan *quotient* yang diberikan adalah '0'. Jika hasil penjumlahan menghasilkan carry out, maka nilai partial remainder yang digunakan untuk penjumlahan selanjutnya adalah nilai hasil penjumlahan tersebut dan *quotient* yang diberikan adalah '1'.
8. Ulangi langkah 5 sampai langkah 7 sebanyak *n* kali hingga didapatkan hasil akhir berupa *n* bit hasil bagi (quotient) dan *n* bit sisa bagi (*remainder*)
9. Selesai.

Perhatikan bahwa terdapat penjumlahan dengan bilangan *2's complement* pada algoritma tersebut. Penjumlahan dengan bilangan *2's complement* tak lain merupakan operasi pengurangan, salah satu ciri dari algoritma jenis pertama.

Setelah memahami bagaimana alur/algoritma dari operasi pembagain yang dilakukan, pengekstraksian ke dalam bentuk blok-blok apa saja yang dibutuhkan dapat dilakukan. Pada subbab selanjutnya akan dijelaskan mengenai perancangan blok-blok dan unit-unit dasar yang dibutuhkan. Perancangan blok-blok dan unit-unit dasar ini dirangkum dalam satu subbab yaitu perancangan *standard cell*.

### 2.2. Perancangan *Standard Cell*

*Cell* dapat diartikan sebagai unit atau blok yang dirancang. Perancangan *standard cell* sendiri dilakukan agar setiap *cell* yang dibuat mempunyai tinggi-lebar yang seragam, lebar VDD-GND yang sama, dan penggunaan metal dengan fungsi yang sama [6]. Perancangan *standar cell* dimulai dari tahap skematik hingga tahap *layout*.

Seperti yang telah dijelaskan sebelumnya, terdapat unit-unit dasar dan blok-blok yang dibutuhkan guna merancang *divider*. Unit-unit dasar dapat berfungsi sebagai penyusun blok maupun digunakan langsung untuk keperluan perancangan *divider*. Unit-unit dasar yang dibutuhkan terdiri atas beberapa gerbang logika dasar, *buffer*, *tristate*, XOR, *multiplexer*, *D flip-flop* tanpa *reset*, dan *D flip-flop* dengan *reset*.

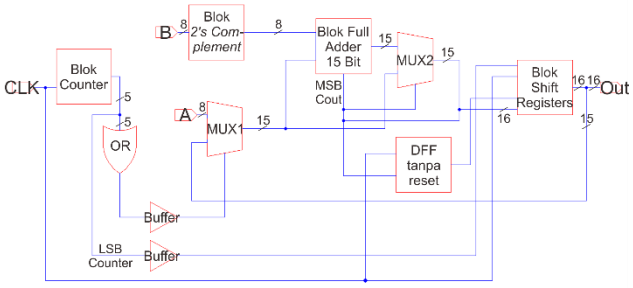
Setelah unit-unit dasar yang dirancang sesuai secara fungsional, maka dilanjutkan dengan perancangan blok. Dari algoritma yang telah dijelaskan pada subbab 2.1, *divider* yang dirancangan memerlukan blok-blok sebagai berikut.

1. Blok *2's complement*, berfungsi untuk mengubah *divisor* menjadi bilangan *2's complement*-nya.
2. Blok *full adder* 15 bit, berfungsi untuk menjumlahkan *partial remainder* dengan bilangan *2's complement* dari *divisor*.
3. Blok *multiplexer*, berfungsi untuk menyeleksi nilai mana yang akan dijumlahkan dengan bilangan *2's complement* dari *divisor* (*dividend* atau *partial remainder*) dan untuk menyeleksi nilai *partial remainder* mana yang akan digeser. Pada *divider* ini, dibutuhkan 2 buah blok *multiplexer* 2 ke 1 sebesar 15 bit.

4. Blok *counter*, berfungsi sebagai kendali masukan dari salah satu blok *multiplexer* dan blok *shift registers*. *Counter* yang diperlukan adalah *counter* yang dapat melakukan perhitungan mulai dari $00000_2$ hingga $10000_2$, karena $10000_2$ merupakan jumlah sinyal *clock* yang dibutuhkan sebuah *divider* untuk menyelesaikan operasi pembagian.
5. Blok *shift registers*, berfungsi untuk mengatur kapan suatu *shift registers* melakukan proses *load* dan kapan melakukan *shift*.

### 2.3. Perancangan *Divider*

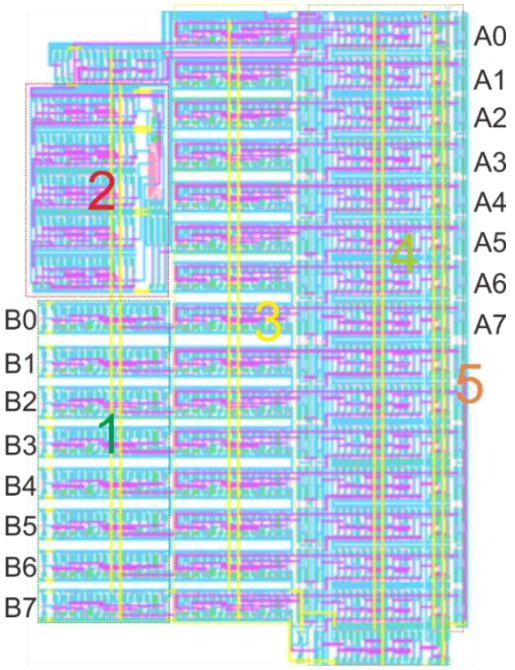
Apabila setiap unit-unit dasar dan blok-blok sudah sesuai secara fungsional, maka langkah selanjutnya adalah melakukan perancangan *divider*. Gambar 1 menunjukkan skematik dari *divider* yang dirancang.




**Gambar 1. Skematik dari *divider***

*Divider* yang dirancang memiliki masukan berupa bilangan yang dibagi (*dividend*) sebesar 8 bit, pembagi (*divisor*) sebesar 8 bit, dan sinyal *clock*. Pada Gambar 1, *dividend* dinotasikan oleh A, *divisor* dinotasikan oleh B, dan sinyal *clock* dinotasikan oleh CLK. Untuk keluarannya, terdiri atas 16 bit bilangan yang terdiri atas hasil bagi (*quotient*) dan sisa bagi (*remainder*). Pada Gambar 1, keluaran dinotasikan oleh Out. Misal Out0 adalah *least significant bit* (LSB) dan Out16 adalah *most significant bit* (MSB), maka Out0 hingga Out7 merupakan *quotient* sementara Out9 hingga Out15 merupakan *remainder*.

Setelah perancangan tahap skematik selesai, langkah selanjutnya adalah perancangan pada tahap *layout*. Gambar 2 menunjukkan *layout* dari *divider* yang dirancang. A0 hingga A7 merupakan *dividend* sementara B0 hingga B7 merupakan *divisor*. Daerah 1 hingga daerah 5 merupakan blok-blok penyusun *divider*. Daerah 1 merupakan blok *2's complement*, daerah 2 merupakan blok *counter*, daerah 3 merupakan blok *full adder* 15 bit, daerah 4 merupakan blok *shift registers*, dan daerah 5 merupakan salah satu dari blok *multiplexer*.




**Gambar 2. Layout dari *divider***

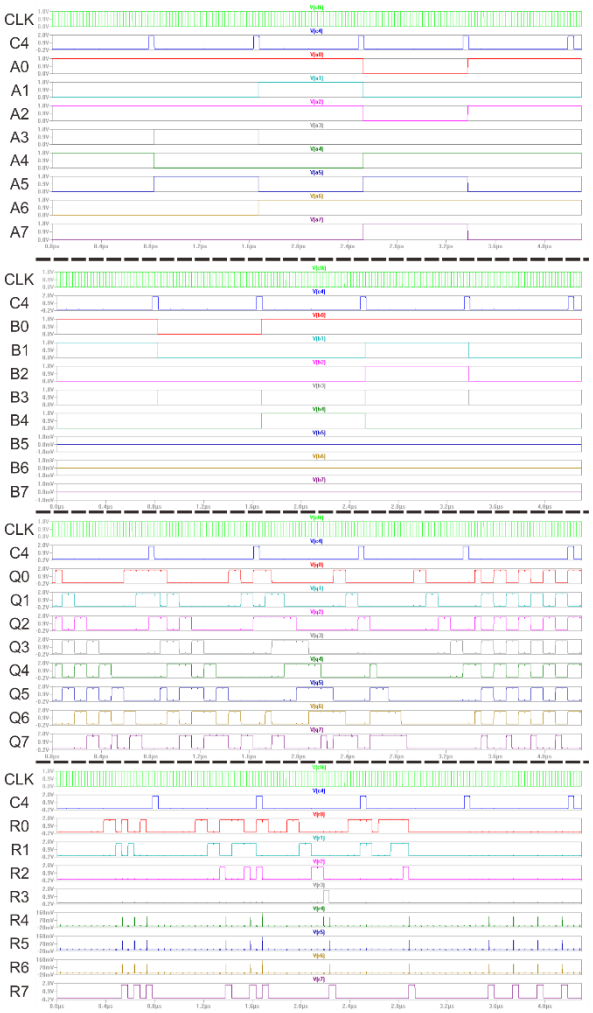
## 3. Hasil dan Analisa

Setelah perancangan *layout* dari *divider* selesai dilakukan, tahap selanjutnya adalah memberikan node masukan dan memberikan *spice code* pada *layout*. Setelah selesai, dilanjutkan dengan melakukan pengujian fungsional, penghitungan jumlah transistor, pengukuran luas, *area coverage*, dan *timing delay*. Untuk melakukan pengujian fungsional dan pengukuran *timing delay*, *layout* dari *divider* diekstrak menggunakan perangkat lunak LTspice.

### 3.1. Pengujian Fungsional

Karena *divider* yang dirancang memiliki besar 8 bit, maka terdapat 65.280 kemungkinan percobaan yang dapat dilakukan (dengan menganggap pembagain dengan bilangan 0 tidak terjadi). Namun pada pengujian fungsional ini, hanya dilakukan 40 percobaan dengan data yang berbeda (40 *dividend* dan 40 *divisor*). Hal ini dikarenakan 40 percobaan yang dilakukan sudah cukup memenuhi variasi percobaan yang diperlukan, misalnya seperti pembagian genap dengan ganjil, pembagian bilangan yang lebih kecil dengan yang lebih besar, pembagian yang menghasilkan hasil bagi bernilai besar namun sisa bagi bernilai kecil atau sebaliknya, dan variasi lainnya.

Empat puluh data yang diambil ini kemudian dibagi menjadi 8 tahap, sehingga setiap tahap terdiri atas 5 data. Gambar 3 menunjukkan gelombang keluaran dari salah satu tahap yang dilakukan, yaitu tahap 1.




**Gambar 3. Pengujian *divider* tahap 1**

Pada Gambar 3, CLK mewakili sinyal *clock*, C4 mewakili MSB dari *counter*, A0 hingga A7 mewakili *dividend*, B0 hingga B7 mewakili *divisor*, Q0 hingga Q7 mewakili *quotient*, dan R0 hingga R7 mewakili *remainder*. Perhatikan bahwa pada A, B, Q, dan R, angka 0 menunjukkan bahwa bilangan tersebut adalah LSB serta angka 7 menunjukkan bahwa bilangan tersebut adalah MSB.

Seperti yang telah dijelaskan pada subbab 2.2, *divider* yang dirancang membutuhkan sinyal *clock* sebanyak $10000_2$ untuk menyelesaikan operasi yang dilakukan. Oleh karena itu, MSB dari *counter* berfungsi sebagai sinyal penanda untuk melihat apakah proses yang dilakukan sudah benar secara fungsional atau belum. Singkatnya, hasil keluaran (*quotient* dan *remainder*) dapat dilihat ketika C4 bernilai 1. Sementara ketika C4 bernilai 0, keluaran yang dihasilkan dapat diabaikan karena keluaran tersebut merupakan hasil pembagian yang belum selesai.
Sebagai contoh, pada tahap 1 ini *dividend* awal yang diberikan adalah $00010101_2$ dan *divisor* akhir yang diberikan adalah $00000011_2$. Ketika C4 bernilai 1, dapat dilihat bahwa keluaran *quotient* bernilai $00000111_2$ dan keluaran *remainder* bernilai $00000000_2$. Hal ini menunjukkan operasi yang dilakukan sudah sesuai secara fungional.

Tabel 1 menunjukkan hasil pengujian fungsional pada setiap tahap. Dari Tabel 1, dapat dilihat bahwa *divider* yang dirancang sudah sesuai secara fungsional.

**Tabel 1. Pengujian fungsional *divider***

| No. | Masukan | | Keluaran | |
|---|---|---|---|---|
| | *Dividend* | *Divisor* | *Quotient* | *Remainder* |
| 1. | 00010101 | 00000011 | 00000111 | 00000000 |
| 2. | 00101101 | 00001000 | 00000101 | 00000101 |
| 3. | 01000111 | 00010001 | 00000100 | 00000011 |
| 4. | 11110000 | 00001111 | 00010000 | 00000000 |
| 5. | 01010101 | 00000001 | 01010101 | 00000000 |
| 6. | 11111111 | 00000010 | 01111111 | 00000001 |
| 7. | 10101010 | 01111101 | 00000001 | 00101101 |
| 8. | 11011100 | 01100100 | 00000010 | 00010100 |
| 9. | 00000011 | 10010011 | 00000000 | 00000011 |
| 10. | 10111110 | 01010000 | 00000010 | 00011110 |
| 11. | 11111111 | 00000001 | 11111111 | 00000000 |
| 12. | 01111000 | 00110011 | 00000010 | 00010010 |
| 13. | 01100011 | 01100010 | 00000001 | 00000001 |
| 14. | 01100100 | 11100111 | 00000000 | 01100100 |
| 15. | 01011001 | 00100101 | 00000010 | 00001111 |
| 16. | 00000001 | 11111111 | 00000000 | 00000001 |
| 17. | 00001111 | 00001010 | 00000001 | 00000101 |
| 18. | 11101000 | 00000100 | 00111010 | 00000000 |
| 19. | 11111010 | 10010110 | 00000001 | 01100100 |
| 20. | 00000000 | 11111111 | 00000000 | 00000000 |
| 21. | 10010101 | 00000111 | 00010101 | 00000010 |
| 22. | 10110101 | 01101111 | 00000001 | 01000110 |
| 23. | 11111010 | 00011001 | 00001010 | 00000000 |
| 24. | 11001000 | 11010011 | 00000000 | 11001000 |
| 25. | 11000110 | 00000011 | 01000010 | 00000000 |
| 26. | 00011001 | 00001011 | 00000010 | 00000011 |
| 27. | 00001010 | 10011101 | 00000000 | 00001010 |
| 28. | 10111010 | 01100101 | 00000001 | 01010101 |
| 29. | 11111110 | 11111111 | 00000000 | 11111110 |
| 30. | 10100101 | 00000101 | 00100001 | 00000000 |
| 31. | 11001010 | 00000110 | 00100001 | 00000100 |
| 32. | 01010111 | 10101000 | 00000000 | 01010111 |
| 33. | 10100110 | 00001010 | 00010000 | 00000110 |
| 34. | 01011011 | 01011011 | 00000001 | 00000000 |
| 35. | 10011000 | 00111000 | 00000010 | 00101000 |
| 36. | 00001001 | 00101111 | 00000000 | 00001001 |
| 37. | 00100001 | 00000011 | 00001011 | 00000000 |
| 38. | 11111001 | 01001101 | 00000011 | 00010010 |
| 39. | 01110000 | 00111011 | 00000001 | 00110101 |
| 40. | 01000011 | 01100100 | 00000000 | 01000011 |

### 3.2. Pengukuran Luas dan *Area Coverage*

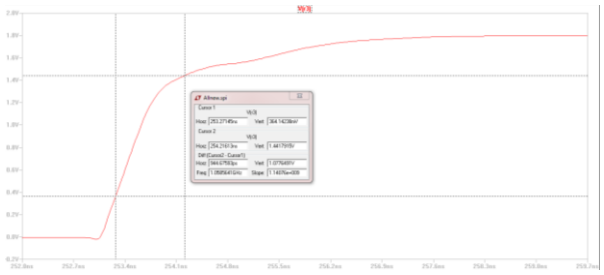
Untuk penghitungan luas dan *area coverage* dari *divider* yang dirancang, dapat digunakan fitur 'Check Area Coverage' pada perangkat lunak Electric. Dari hasil pengukuran, didapati bahwa *divider* yang dirancang memiliki dimensi sebesar 1488 x 2240 $\lambda^2$. Karena teknologi yang digunakan adalah 180nm, maka besarnya $\lambda$ adalah 90nm. Dengan demikian, luas dari *divider* yang dirancang adalah sebesar 0,027mm$^2$.

Setelah pengukuran luas, langkah selanjutnya adalah menghitung *area coverage*. *Area coverage* dapat diartikan sebagai seberapa rapat daerah yang digunakan dalam perancangan. Semakin tinggi nilai *area coverage*-nya, maka rancangan tersebut semakin baik karena susunannya semakin rapat sehingga tidak ada daerah yang kosong atau terbuang. Pada perancangan *divider* ini, didapati besar *area coverage*-nya adalah 45,875%.

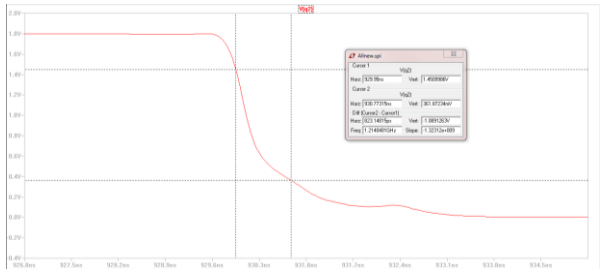
### 3.3. Pengukuran *Timing Delay*

Pengukuran *timing delay* dilakukan pada gelombang keluaran hasil akhir pada setiap variasi masukan di setiap tahap. *Timing delay* yang diukur meliputi *rise time* ($T_r$), *fall time* ($T_f$), *propagation delay rise time* ($T_{pdr}$), *propagation delay fall time* ($T_{pdf}$), dan *propagation delay time* ($T_{pd}$).

Tabel 2 menunjukkan pengukuran *propagation delay time* yang dilakukan. Dari Tabel 2, dapat dilihat bahwa *propagation delay time* terbesar adalah 3,644ns (masukan nomor 4). Setiap data yang diambil diukur pada seluruh variasi masukan di setiap tahap dengan mencari keluaran mana yang memiliki *delay* paling besar. Gambar 4 dan Gambar 5 beruturut.-turut menunjukkan pengukuran $T_r$ dan $T_f$ terbesar, sementara Gambar 6 dan Gambar 7 berturut-turut menunjukkan pengukuran $T_{pd}$ dan $T_{pdf}$ yang menghasilkan nilai $T_{pd}$ dengan nilai terbesar.



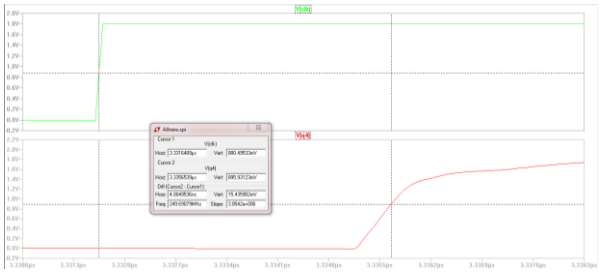
**Gambar 4. Pengukuran $T_r$ dari *divider***



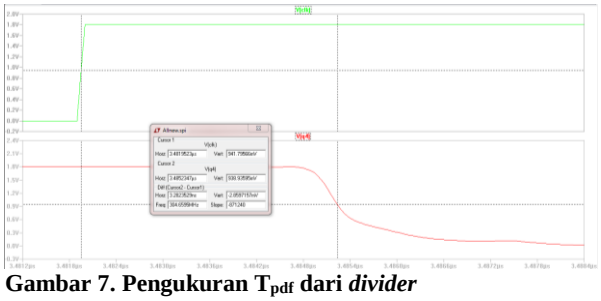
**Gambar 5. Pengukuran $T_f$ dari *divider***

**Tabel 2. Pengukuran *time propagation delay***

| No. | Masukan | | $T_{pd}$ |
|---|---|---|---|
| | *Dividend* | *Divisor* | (ns) |
| 1. | 00010101 | 00000011 | 0,628 |
| 2. | 00101101 | 00001000 | 0,926 |
| 3. | 01000111 | 00010001 | 2,543 |
| 4. | 11110000 | 00001111 | 3,644 |
| 5. | 01010101 | 00000001 | 1,967 |
| 6. | 11111111 | 00000010 | 2,658 |
| 7. | 10101010 | 01111101 | 2,533 |
| 8. | 11011100 | 01100100 | 2,629 |
| 9. | 00000011 | 10010011 | 2,544 |
| 10. | 10111110 | 01010000 | 2,525 |
| 11. | 11111111 | 00000001 | 1,928 |
| 12. | 01111000 | 00110011 | 2,621 |
| 13. | 01100011 | 01100010 | 1,969 |
| 14. | 01100100 | 11100111 | 2,547 |
| 15. | 01011001 | 00100101 | 2,266 |
| 16. | 00000001 | 11111111 | 2,626 |
| 17. | 00001111 | 00001010 | 2,699 |
| 18. | 11101000 | 00000100 | 3,538 |
| 19. | 11111010 | 10010110 | 2,541 |
| 20. | 00000000 | 11111111 | 2,624 |
| 21. | 10010101 | 00000111 | 1,965 |
| 22. | 10110101 | 01101111 | 2,687 |
| 23. | 11111010 | 00011001 | 3,621 |
| 24. | 11001000 | 11010011 | 2,612 |
| 25. | 11000110 | 00000011 | 3,623 |
| 26. | 00011001 | 00001011 | 2,550 |
| 27. | 00001010 | 10011101 | 2,624 |
| 28. | 10111010 | 01100101 | 2,630 |
| 29. | 11111110 | 11111111 | 2,296 |
| 30. | 10100101 | 00000101 | 3,620 |
| 31. | 11001010 | 00000110 | 2,697 |
| 32. | 01010111 | 10101000 | 2,687 |
| 33. | 10100110 | 00001010 | 2,568 |
| 34. | 01011011 | 01011011 | 1,970 |
| 35. | 10011000 | 00111000 | 2,702 |
| 36. | 00001001 | 00101111 | 2,636 |
| 37. | 00100001 | 00000011 | 1,962 |
| 38. | 11111001 | 01001101 | 2,621 |
| 39. | 01110000 | 00111011 | 2,547 |
| 40. | 01000011 | 01100100 | 2,698 |



**Gambar 6. Pengukuran $T_{pdr}$ dari *divider***



**Gambar 7. Pengukuran $T_{pdf}$ dari *divider***

Dari hasil pengukuran dengan nilai terbesar yang dilakukan, didapatkan $T_r$ sebesar 0,945ns, $T_f$ sebesar 0,823ns, $T_{odr}$ sebesar 4,005ns, dan $T_{pdf}$ sebesar 3,282ns. Tabel 3 merangkum hasil pengukuran *timing delay* maksimal yang dilakukan pada *divider*.

**Tabel 3. Pengukuran *timing delay* maksimal**

| $T_r$ (ns) | $T_f$ (ns) | $T_{pdr}$ (ns) | $T_{pdf}$ (ns) | $T_{pd}$ (ns) |
|---|---|---|---|---|
| 0,945 | 0,823 | 4,005 | 3,282 | 3,644 |

### 3.4. Analisa

Pada subbab 2.3 telah dijelaskan bahwa selain *dividend* dan *divisor*, *divider* yang dirancang memiliki masukan sinyal *clock*. Besarnya sinyal *clock* sangat penting dalam rancangan sekuensial. Apabila sinyal *clock* yang diberikan terlalu kecil periodenya, maka ada kemungkinan nilai yang diinginkan menjadi tidak sesuai. Apabila sinyal *clock* yang diberikan terlalu besar periodenya, maka waktu yang dibutuhkan untuk menyelesaikan suatu proses menjadi lebih lama. Untuk itu, diperlukan sinyal *clock* dengan periode atau frekuensi yang aman agar proses yang dilakukan menghasilkan keluaran yang sesuai, namun juga tidak memakan waktu yang lama.

Pada pengujian yang dilakukan, sinyal *clock* yang digunakan adalah sebesar 20MHz atau 50ns untuk 1 periode gelombang. Pemilihan besar sinyal *clock* tersebut dikarenakan sinyal *clock* ini menghasilkan keluaran yang sesuai dan hasil tersebut tidak berubah ketika memang belum waktunya untuk berubah, khususnya jika pengujian yang dilakukan adalah sebanyak 5 variasi masukan. Dari pengujian fungsional yang dilakukan dan *propagation delay time* yang dihasilkan, sinyal *clock* ini dapat diperkecil hingga memiiki periode sebesar 16ns. Namun, penggunaan sinyal *clock* sebesar 16ns memiliki risiko keluaran bergeser sebelum waktunya yang lebih besar.

Dengan menggunakan sinyal *clock* sebesar 50ns, maka proses yang dibutuhkan suatu *divider* untuk menyelesaikan perhitungan dalam satu siklus adalah sebesar 50ns x 16, yakni 800ns. Enam belas merupakan banyaknya sinyal *clock* yang dibutuhkan oleh *divider* ini untuk menyelesaikan perhitungan pada satu siklus.

Penggunaan sinyal *clock* sebesar 20MHz sendiri bersifat tidak mutlak. Apabila rancangan yang dibuat memiliki luas dan *timing delay* yang berbeda, bisa saja sinyal *clock* sebesar 20MHz tidak dapat digunakan karena menghasilkan keluaran yang kurang sesuai. Dengan kata lain, pemilihan besar sinyal *clock* berkaitan dengan luas dan *timing delay* yang dihasilkan.

## 4. Kesimpulan

Dari pengujian yang telah dilakukan, *divider* yang dirancang pada penelitian ini telah sesuai secara fungsional. *Divider* yang dirancang memiliki luas sebesar 0,027mm$^2$, *area coverage* sebesar 45,975%, dan *propagation delay* sebesar 3,644ns. *Divider* ini diuji pada dengan sinyal *clock* berfrekuensi 20MHz. Untuk pengembangan selanjutnya, dapat dirancang suatu *divider* dengan teknologi yang berbeda, sehingga menghasilkan luas dan *propagation delay time* yang berbeda untuk dibandingkan lebih jauh seperti apa perbedaannya.

## Referensi


[1]. P. Nair, D. Kudithipudi, and E. John, “Design and Implementation of a CMOS Non-Restoring Divider,” pp. 211–217, 2006.

[2]. S. K. Panda, “A Novel Vedic Divider Architecture with Reduced Delay for VLSI Applications,” vol. 120, no. 17, pp. 31–36, 2015.

[3]. C.-L. Wey, “Design of A Fast Radix-4 SRT Divider and its VLSI Implementation,” vol. 146, pp. 205–210, 1999.

[4]. J. Melorose, R. Perroy, and S. Careas, “Division,” *Statew. Agric. L. Use Baseline 2015*, vol. 1, 2015.

[5]. M. M. Mano and C. Kime, *Logic and Computer Desing Fundamentals*. 2014.

[6]. N. E. H. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, vol. 53, no. 9. 2013.