IMPLEMENTASI CORDIC DI FPGA DENGAN PARALELISASI PERHITUNGAN ITERASI

untuk memenuhi salah satu persyaratan

mencapai derajat Sarjana S1

Disusun oleh:

Muhammad Farid Akbar

16524090

Jurusan Teknik Elektro

Fakultas Teknologi Industri

Universitas Islam Indonesia

Yogyakarta

2020

LEMBAR PENGESAHAN

IMPLEMENTASI CORDIC DI FPGA DENGAN PARALELISASI PERHITUNGAN ITERASI

TUGAS AKHIR

Diajukan sebagai Salah Satu Syarat untuk Memperoleh
Gelar Sarjana Teknik
pada Program Studi Teknik Elektro
Fakultas Teknologi Industri
Universitas Islam Indonesia

Disusun oleh:

Muhammad Farid Akbar
16524090

Yogyakarta, 6 April 2020

Menyetujui,

Pembimbing

Dr. Eng., Hendra Setiawan, S.T., M.T.
025200526

LEMBAR PENGESAHAN

SKRIPSI

IMPLEMENTASI CORDIC DI FPGA DENGAN PARALELISASI PERHITUNGAN ITERASI

Dipersiapkan dan disusun oleh:

Muhammad Farid Akbar

16524090

Telah dipertahankan di depan dewan penguji

Pada tanggal: 11 Mei 2020

Susunan dewan penguji

Ketua Penguji : Dr. Eng., Hendra Setiawan, S.T., M.T., _____

Anggota Penguji 1: Alvin Sahroni, S.T., M.Eng., Ph.D., _____

Anggota Penguji 2: Yusuf Aziz Amrulloh, S.T., M.Eng., Ph.D., _____

Skripsi ini telah diterima sebagai salah satu persyaratan untuk memperoleh gelar Sarjana

Tanggal: 20 Mei 2020

Ketua Program Studi Teknik Elektro

Yusuf Aziz Amrulloh, S.T., M.Eng., Ph.D.

045240101

PERNYATAAN

Dengan ini Saya menyatakan bahwa:

1. Skripsi ini tidak mengandung karya yang diajukan untuk memperoleh gelar kesarjanaan di suatu Perguruan Tinggi, dan sepanjang pengetahuan Saya juga tidak mengandung karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.
2. Informasi dan materi Skripsi yang terkait hak milik, hak intelektual, dan paten merupakan milik bersama antara tiga pihak yaitu penulis, dosen pembimbing, dan Universitas Islam Indonesia. Dalam hal penggunaan informasi dan materi Skripsi terkait paten maka akan diskusikan lebih lanjut untuk mendapatkan persetujuan dari ketiga pihak tersebut diatas.

Yogyakarta, 6 April 2020

Muhammad Farid Akbar

KATA PENGANTAR

Dengan menyebut nama Allah Yang Maha Pengasih lagi Maha Penyayang. Segala puji bagi Allah, Tuhan semesta alam. Tiada daya dan upaya kecuali dengan pertolongan Allah Yang Maha Tinggi dan Maha Agung. Semoga sholawat serta salam selalu tercurahkan kepada Rasulullah Muhammad SAW, para keluarga, sahabat, dan tabi'in, serta umat beliau hingga hari akhir. Aamiin.

Atas izin Allah SWT, laporan tugas akhir yang berjudul "IMPLEMENTASI CORDIC DI FPGA DENGAN PARALELISASI PERHITUNGAN ITERASI" ini dapat diselesaikan dengan baik. Laporan ini disusun untuk memenuhi salah satu syarat untuk menyelesaikan studi jenjang pendidikan S1 di jurusan Teknik Elektro Universitas Islam Indonesia.

Selama mengikuti kegiatan pembelajaran di lingkungan Universitas Islam Indonesia hingga penyelesaian tugas akhir ini, penulis ingin mengucapkan terima kasih kepada pihak-pihak yang telah memberikan bantuan dan dukungan kepada penulis, khususnya kepada:

1. Kedua orang tua dan keluarga yang telah memberikan doa, semangat, serta dukungan moral dan material dalam menjalankan studi di Universitas Islam Indonesia.
2. Bapak Yusuf Aziz Amrullah, S.T., M.Eng., Ph.D. selaku Ketua Jurusan Teknik Elektro Universitas Islam Indonesia.
3. Bapak Dr. Eng., Hendra Setiawan, S.T., M.T. selaku Dosen Pembimbing tugas akhir yang telah mendampingi dan memberikan masukan dalam pengerjaan tugas akhir ini.
4. Seluruh dosen dan staff Teknik Elektro Universitas Islam Indonesia yang telah memberikan ilmu serta membantu penulis selama mengikuti kegiatan pembelajaran di Jurusan Teknik Elektro Universitas Islam Indonesia.
5. Rekan-rekan mahasiswa Teknik Elektro Universitas Islam Indonesia, khususnya angkatan 2016 yang telah berjuang bersama dari awal hingga akhir perkuliahan.
6. Serta seluruh pihak yang terlibat selama masa perkuliahan di Universitas Islam Indonesia yang tidak bisa disebutkan satu per satu.

Penulis menyadari bahwa laporan tugas akhir ini masih jauh dari kata sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun sebagai acuan untuk memperbaiki penulisan yang akan datang. Semoga laporan ini dapat bermanfaat bagi pembaca.

Yogyakarta, 6 April 2020

Muhammad Farid Akbar

ARTI LAMBANG DAN SINGKATAN

CORDIC	: *Coordinate Rotation Digital Computer*
LUT	: *Look Up Table*
FPGA	: *Field-Programmable Gate Array*
HDL	: *Hardware Description Language*
VHDL	: *Very High Speed Integrated Circuit Hardware Description Language*
MHz	: *Megahertz*
ns	: *nanosecond*

ABSTRAK

Algoritme CORDIC merupakan salah satu algoritme yang paling populer untuk menyelesaikan permasalahan trigonometri. Dalam praktiknya, algoritme ini membutuhkan proses iterasi yang panjang untuk mendapatkan hasil yang akurat. Pada penelitian ini, perhitungan iterasi algoritme CORDIC dilakukan secara paralel dengan maksud untuk mempercepat proses perhitungan iterasi. Tujuan dari penelitian ini untuk mengetahui pengaruh paralelisasi komputasi iterasi algoritme CORDIC terhadap *latency* yang dihasilkan. Proses paralelisasi dilakukan dengan mengalikan matriks-matriks algoritme CORDIC secara bersamaan untuk memperoleh *output* cos dan sin pada sudut dengan resolusi 0,75˚±5% pada 16 iterasi algoritme CORDIC, sehingga terdapat 4 tingkat perkalian matriks. Untuk menghemat jumlah *resource* yang digunakan, sistem ini dirancang untuk langsung melakukan perkalian matriks tingkat 2. Perkalian matriks dilakukan menggunakan *decoder* 4-bit untuk perkalian tingkat 2 dan blok *multiplier* 9-bit untuk perkalian tingkat 3 dan 4. Proses verifikasi sistem ini dilakukan dalam dua tahap, yaitu verifikasi fungsional dengan aplikasi ModelSim-Altera serta verifikasi *hardware* menggunakan modul FPGA Cyclone IV EP4CE6E228N. Hasil pengujian menunjukkan bahwa *output* trigonometri dari sistem ini memiliki akurasi lebih dari 95% untuk sudut dengan resolusi 0,0625˚. Sistem ini membutuhkan 3336 *logic elements*, 21 *multiplier* 9-bit, dan waktu *latency* 596,1 ns. Dibandingkan dengan penelitian lain, jumlah LUT dan waktu *latency* yang dihasilkan oleh sistem ini tidak mengalami peningkatan yang cukup signifikan. Hal ini dikarenakan sistem ini masih membutuhkan proses iterasi serial untuk mendapatkan arah rotasi CORDIC serta menggunakan resolusi bit *output* yang jauh lebih besar.

Kata kunci: CORDIC, paralelisasi, cos, sin, *multiplier*, LUT, *latency*, FPGA

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 Latar Belakang Masalah

Di era modern ini, sistem digital dapat ditemui hampir di semua aspek kehidupan. Untuk dapat mengendalikan sistem digital tersebut, diperlukan suatu sistem kendali digital. Salah satu aplikasi yang menggunakan sistem kendali digital adalah sistem kinematika *manipulator* yang menggunakan persamaan trigonometri. Implementasi trigonometri pada pengendali sistem digital tidak mudah untuk dilakukan. Implementasi persamaan trigonometri dapat dilakukan menggunakan metode *look up table* (LUT), namun metode tersebut memerlukan kapasitas penyimpanan yang besar untuk tingkat presisi yang lebih tinggi [1]. Sehingga pada tahun 1959, Jack Volder memperkenalkan sebuah algoritme menggunakan konsep rotasi untuk sistem digital yang dinamakan algoritme *Coordinat Rotation Digital Computer* (CORDIC) [2].

Algoritme CORDIC menggunakan metode iterasi dari rotasi sudut bebas untuk memperoleh nilai vektor atau sudut akhir yang diinginkan [2]. Karena menggunakan metode iterasi, algoritme ini membutuhkan jumlah iterasi yang banyak untuk mendapatkan hasil yang akurat. Sebagai contoh, *stepper motor* PFCU25-24D1G memiliki sudut *step* sebesar 0,75˚±5% [3].

Banyaknya iterasi akan menyebabkan perhitungan menjadi lebih lama dan membutuhkan memori yang lebih besar. Seperti hasil yang diperoleh dari penelitian [4]. Pada penelitian tersebut diperoleh korelasi antara tingkat akurasi hasil dengan panjang iterasi yang harus dilakukan. Penelitian [5] menunjukkan bahwa untuk mendapatkan hasil dengan tingkat presisi yang tinggi diperlukan *latency* yang lebih lama.

Proses iterasi dapat dipercepat dengan proses paralelisasi. Pada proses paralelisasi, proses komputasi yang seharusnya bersifat sekuensial dapat dilakukan secara paralel sehingga perhitungan dapat menjadi lebih cepat. Pada penelitian ini, akan dilakukan modifikasi komputasi iterasi dari algoritme CORDIC dengan paralelisasi sehingga perhitungan dapat dilakukan lebih cepat.

1.2 Rumusan Masalah

1. Bagaimana melakukan paralelisasi komputasi iterasi pada algoritme CORDIC dan mengimplementasikannya menggunakan FPGA?

2. Bagaimana pengaruh paralelisasi komputasi iterasi pada algoritme CORDIC terhadap *latency* yang dihasilkan?

1.3 Batasan Masalah

1. Algoritme CORDIC digunakan untuk perhitungan cos dan sin.
2. Sistem ini dirancang dengan asumsi untuk digunakan pada kontrol *stepper motor* dengan kecepatan maksimum 3000 rpm.
3. Implementasi *hardware* menggunakan FPGA Cyclone IV EP4CE6E22C8N *Development Board*.
4. *Resource* yang terlibat dalam perancangan adalah *logic element*, *register*, dan modul *multiplier*.

1.4 Tujuan Penelitian

1. Merancang teknik paralelisasi komputasi iterasi pada algoritme CORDIC dan mengimplementasikannya menggunakan FPGA.
2. Mengetahui pengaruh paralelisasi komputasi iterasi pada algoritme CORDIC terhadap *latency* yang dihasilkan.

1.5 Manfaat Penelitian

1. Menjadi salah satu referensi untuk implementasi kinematika *manipulator* dengan perangkat FPGA.
2. Menambah khazanah keilmuan di bidang *embedded system*.

BAB 2
TINJAUAN PUSTAKA

2.1 Studi Literatur

Saha, dkk membahas tentang implementasi FPGA terhadap fungsi *arcsin* menggunakan algoritme CORDIC [4]. Riset ini dilakukan untuk memperoleh tingkat presisi yang tinggi dalam perhitungan fungsi *arcsin*. Penggunaan algoritme CORDIC dalam riset ini karena algoritme CORDIC menyediakan metode yang efektif untuk menghitung sejumlah besar fungsi transendental karena hanya membutuhkan operasi penjumlahan, pengurangan, dan pergeseran. Metode yang digunakan pada algoritme CORDIC yaitu dengan memanfaatkan fungsi rotasi dari *plane* 2 dimensi dan mengasumsikan nilai $\tan(\alpha) = 2^{-i}$, dengan i adalah nomor iterasi, kemudian substitusi nilai tan(α) pada fungsi rotasi dan dilakukan iterasi hingga jumlah iterasi yang telah ditentukan. Hasil riset didapatkan bahwa lamanya komputasi dan tingkat akurasi bergantung pada banyaknya iterasi. Pada iterasi ke-8 diperoleh siklus *clock* sebanyak 26 siklus dengan *error* 0,51%. Pada iterasi ke-9 dan ke-10 jumlah siklus *clock* menjadi lebih banyak, masing-masing 29 dan 32 siklus, namun *error* yang dihasilkan menjadi lebih kecil, yaitu 0,13% dan 0,06%. Jumlah iterasi yang lebih banyak menyebabkan tingkat akurasi menjadi lebih tinggi, namun waktu komputasi menjadi lebih lama. Dengan dilakukannya riset ini, dapat diketahui bahwa algoritme CORDIC dapat digunakan pada berbagai aplikasi karena algoritme CORDIC dapat dimodifikasi dengan mudah untuk beroperasi pada tingkat akurasi yang lebih tinggi, tergantung pada kebutuhan aplikasi. Namun, karena algoritme CORDIC bersifat iteratif, maka perlu jumlah iterasi yang besar untuk memperoleh hasil yang presisi sehingga dapat menyebabkan waktu komputasi menjadi lebih lama dan dibutuhkan kapasitas penyimpanan yang lebih besar pula.

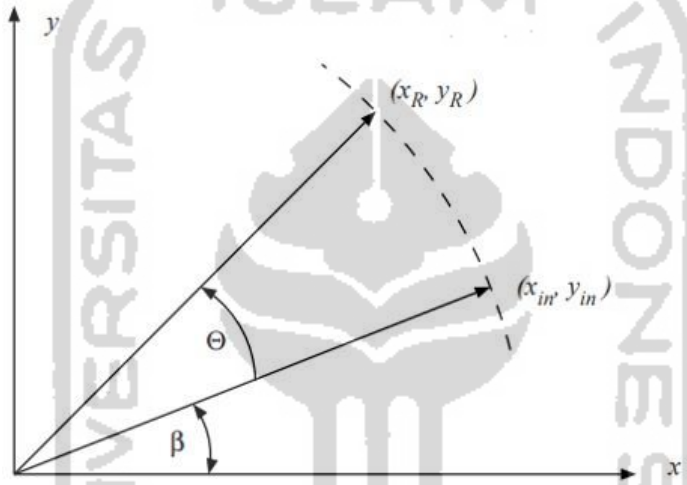
Zhang, dkk membahas tentang akurasi tinggi dalam implementasi fungsi trigonometri pada FPGA menggunakan metode ekspansi *Taylor* [6]. Alasan dilakukannya riset ini yaitu untuk mengetahui apakah metode ekspansi *Taylor* dapat digunakan untuk menghitung fungsi trigonometri serta bagaimana tingkat akurasi yang dihasilkan oleh metode ini. Riset ini bertujuan untuk mendapatkan hasil dengan akurasi tinggi terhadap perhitungan fungsi trigonometri dengan menggunakan metode ekspansi *Taylor*. Cara yang diusulkan dalam makalah ini yaitu dengan melakukan ekspansi *Taylor* di beberapa subbagian rata-rata berdasarkan analisis kesalahan sisa *Lagrange* dan melakukan ekspansi *Taylor* di pusat setiap bagian. Hasil penelitian menunjukkan bahwa semakin besar jumlah data sumber, akan mempercepat waktu kalkulasi rata-rata pada setiap data. Ketika banyak data mendekati tak hingga, periode kalkulasi rata-rata menjadi 5,88 siklus. Kelebihan dari metode ekspansi *Taylor* ini yaitu metode ini memiliki *error* yang sangat kecil,

hingga mencapai nilai 10^{-6}. Selain itu metode ini dapat digunakan untuk menghitung nilai dari fungsi trigonometri pada sudut berapapun.

2.2 Tinjauan Teori

2.2.1 Algoritme CORDIC

Algoritme CORDIC merupakan sebuah algoritme yang menggunakan metode perulangan (iterasi) rotasi untuk menghitung berbagai fungsi dasar [2]. Misalkan kita memiliki sistem yang efisien yang menerima vektor dan memutarnya dengan sudut sebesar Θ, maka sistem tersebut dapat diilustrasikan oleh Gambar 2.1.



Gambar 2.1 Rotasi Vektor *Input* Sebesar Θ [7]

Pada Gambar 2.1, x_{in}, y_{in}, dan β merupakan posisi awal dari sistem serta x_R dan y_R adalah koordinat akhir sistem. Bentuk matriks rotasi yang menggambarkan hubungan posisi awal dan akhir ditunjukkan oleh Persamaan (2.1).

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \begin{bmatrix} \cos(\Theta) & -\sin(\Theta) \\ \sin(\Theta) & \cos(\Theta) \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} = \begin{bmatrix} x_{in}\cos(\Theta) - y_{in}\sin(\Theta) \\ x_{in}\sin(\Theta) + y_{in}\cos(\Theta) \end{bmatrix} \quad (2.1)$$

x_R = koordinat akhir pada sumbu x

y_R = koordinat akhir pada sumbu y

Θ = sudut putar

x_{in} = koordinat awal pada sumbu x

y_{in} = koordinat awal pada sumbu y

Persamaan (2.1) dapat disederhanakan menjadi Persamaan (2.2).

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \cos(\Theta) \begin{bmatrix} 1 & -\tan(\Theta) \\ \tan(\Theta) & 1 \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \quad (2.2)$$

x_R = koordinat akhir pada sumbu x

y_R = koordinat akhir pada sumbu y

Θ = sudut putar

x_{in} = koordinat awal pada sumbu x

y_{in} = koordinat awal pada sumbu y

Dari Persamaan (2.2) diketahui bahwa untuk melakukan satu kali rotasi memerlukan empat buah perkalian. Algoritme CORDIC menyederhanakan operasi perkalian menjadi operasi pergeseran sederhana dengan membatasi sudut rotasi pada nilai $\tan(\Theta) = 2^{-i}$, dengan i adalah nomor iterasi [8].

Berdasarkan gagasan di atas, didapatkan hubungan antara iterasi *i* dengan besar sudut putar Θ_i pada Tabel 2.1.

Tabel 2.1 Hubungan Nomor Iterasi (*i*) dan Sudut Iterasi (Θ_i)

Iterasi (*i*)	Tan(Θ) = 2^{-i}	Θ_i (˚)
0	1	45
1	0,5	26,565051
2	0,25	14,036243
3	0,125	7,1250163
4	0,0625	3,5763344
5	0,03125	1,7899106
6	0,015625	0,8951737
7	0,007813	0,4476142
8	0,003906	0,2238105
9	0,001953	0,1119057
10	0,000977	0,0559529
11	0,000488	0,0279765
12	0,000244	0,0139882
13	0,000122	0,0069941
14	0,000061	0,0034971
15	0,000031	0,0017485

Dengan mensubstitusikan nilai $\tan(\Theta) = 2^{-i}$ pada Persamaan (2.2) dan melakukan rotasi secara berulang, maka persamaan rotasi algoritme CORDIC dapat dinyatakan sebagai Persamaan (2.3). Dengan α_i menyatakan arah rotasi setiap iterasi, yang bernilai +1 jika sudut iterasi berputar berlawanan arah jarum jam dan bernilai -1 jika sudut iterasi berputar searah jarum jam.

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \left(\prod_{i=0}^{n-1} \cos(\Theta_i) \right) \left(\prod_{i=0}^{n-1} \begin{bmatrix} 1 & -\alpha_i \times 2^{-i} \\ \alpha_i \times 2^{-i} & 1 \end{bmatrix} \right) \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \quad (2.3)$$

x_R = koordinat akhir pada sumbu x

y_R = koordinat akhir pada sumbu y

Θ = sudut putar

n = banyak iterasi

α = arah rotasi

x_{in} = koordinat awal pada sumbu x

y_{in} = koordinat awal pada sumbu y

Untuk jumlah iterasi sebanyak 16 kali, hasil perkalian $\prod_{i=0}^{15} \cos(\Theta_i)$ menghasilkan nilai 0,6072529. Sehingga Persamaan (2.3) dapat disederhanakan menjadi Persamaan (2.4).

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = 0{,}6072529 \times \left(\prod_{i=0}^{15} \begin{bmatrix} 1 & -\alpha_i \times 2^{-i} \\ \alpha_i \times 2^{-i} & 1 \end{bmatrix} \right) \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \quad (2.4)$$

x_R = koordinat akhir pada sumbu x

y_R = koordinat akhir pada sumbu y

α = arah rotasi

x_{in} = koordinat awal pada sumbu x

y_{in} = koordinat awal pada sumbu y

Jika koordinat awal adalah (1, 0), maka koordinat akhir akan sama dengan nilai cos dan sin sudut target. Sehingga Persamaan (2.5) menjadi *output* trigonometri algoritme CORDIC.

$$\begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} = 0{,}6072529 \times \left(\prod_{i=0}^{15} \begin{bmatrix} 1 & -\alpha_i \times 2^{-i} \\ \alpha_i \times 2^{-i} & 1 \end{bmatrix} \right) \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (2.5)$$

φ = sudut target

α = arah rotasi

2.2.2 *Field Programmable Gate Array*

Field Programmable Gate Array (FPGA) merupakan suatu perangkat yang mengandung sebuah matriks dari sirkuit gerbang logika yang dapat dikonfigurasi ulang [9]. FPGA dapat digunakan hampir di semua area yang membutuhkan sistem digital.

Kelebihan FPGA dibandingkan dengan mikrokontroler, yaitu mikrokontroler hanya dapat menjalankan instruksi secara berurutan sehingga tidak dapat menyelesaikan banyak tugas pemrosesan secara bersamaan. Sedangkan FPGA dapat menjalankan perintah secara paralel sehingga operasi dapat dilakukan lebih cepat [9].

Salah satu jenis FPGA adalah Cyclone IV EP4CE6E22C8N yang diproduksi oleh Altera. Arsitektur dari perangkat ini, yaitu FPGA *Core Fabric* yang terdiri dari *logic elements* (LUT dan register), blok memori, dan *multipliers*; fitur I/O; *Clock Management*; antamuka eksternal; serta skema konfigurasi [10]. *Resource* dari komponen ini terdiri dari memori RAM 270 kb, 6.272 *logic elements*, 15 buah *embedded multiplier* 18×18, dan maksimum 179 PIN I/O [10].

Fitur-fitur yang digunakan dalam penelitian ini, yaitu *logic element*, *multiplier*, *port* reset, *clock*, *seven segment*, *push button*, dan LED.

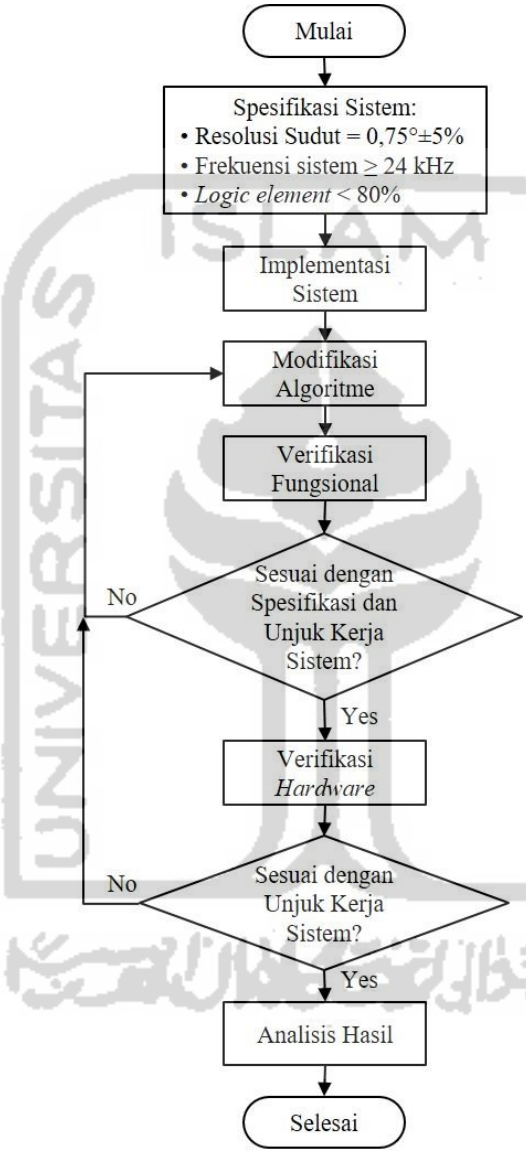
Untuk dapat memprogram FPGA diperlukan suatu bahasa yang disebut *Hardware Description Language* (HDL). Kode HDL digunakan untuk menuliskan sifat, sinyal dan fungsionalitas deskripsi berbasis *hardware* dari suatu rangkaian [11]. *Very High Speed Integrated Circuit* HDL (VHDL) merupakan salah satu jenis bahasa HDL yang dapat digunakan untuk mendeskripsikan berbagai fungsi rangkaian digital [11].

Proses komputasi paralel pada FPGA dilakukan menggunakan *component statement* yang bertujuan untuk menentukan hierarki desain atau membangun *netlist* pada VHDL dengan cara menghubungkan setiap port desain dari *component* ke bentuk sinyal pada *entity* ini [11].

BAB 3
METODOLOGI

3.1 Alur Penelitian

Langkah-langkah penelitian ini dijabarkan pada Gambar 3.1.



Gambar 3.1 Diagram Alir Penelitian

Penelitian ini diawali dari spesifikasi sistem hingga analisis hasil. Proses spesifikasi sistem bertujuan untuk menentukan target yang akan dicapai dalam penelitian ini. Kemudian dilakukan implementasi sistem pada pemrograman VHDL. Selanjutnya dilakukan modifikasi algoritme CORDIC sehingga dapat melakukan komputasi iterasi secara paralel. Lalu dilakukan verifikasi

fungsional serta verifikasi *hardware*. Setelah didapat hasil yang sesuai dengan spesifikasi dan unjuk kerja, sistem dianalisis seberapa besar penurunan *latency* yang dihasilkan.

3.2 Alat dan Bahan

Alat dan bahan yang digunakan dalam penelitian ini yaitu FPGA Cyclone IV EP4CE6E22C8N dan personal komputer yang sudah terpasang aplikasi Quartus Prime dan ModelSim-Altera.

3.3 Spesifikasi Sistem

Sistem ini dirancang dengan spesifikasi sebagai berikut:

1. Komputasi algoritme CORDIC dilakukan secara paralel.
2. Target trigonometri yang akan dihitung dari sistem ini adalah nilai cos dan sin dengan resolusi sudut 0,75˚±5%, menyesuaikan dengan tipe *stepper motor* dengan sudut *step* terkecil pada [3]. Perhitungan sudut menggunakan algortime CORDIC untuk mencapai sudut target ditunjukkan pada Tabel 3.1.

Tabel 3.1 Perhitungan Sudut Resolusi dengan Algoritme CORDIC

Sudut Target (derajat)	Iterasi (i)	Θ_i (derajat)	Tanda Rotasi	Sudut *Output* (derajat)	Selisih Sudut (derajat)	*Error* (%)
0,75	0	45	Positif	45	44,25	5900
	1	26,5650512	Negatif	18,4349488	17,6849488	2357,9932
	2	14,0362435	Negatif	4,3987054	3,6487054	486,4941
	3	7,1250163	Negatif	-2,7263110	3,4763110	463,5081
	4	3,5763344	Positif	0,8500234	0,1000234	13,3365
	5	1,7899106	Negatif	-0,9398872	1,6898872	225,3183
	6	0,8951737	Positif	-0,0447135	0,7947135	105,9618
	7	0,4476142	Positif	0,4029007	0,3470993	46,2799
	8	0,2238105	Positif	0,6267112	0,1232888	16,4385
	9	0,1119057	Positif	0,7386168	0,0113832	1,5178
	10	0,0559529	Positif	0,7945697	0,0445697	5,9426
	11	0,0279765	Negatif	0,7665933	0,0165933	2,2124
	12	0,0139882	Negatif	0,7526050	0,0026050	0,3473
	13	0,0069941	Negatif	0,7456109	0,0043891	0,5852
	14	0,0034971	Positif	0,7491080	0,0008920	0,1189
	15	0,0017485	Positif	0,7508565	0,0008565	0,1142

Dengan nilai selisih sudut dan *error* didapat dari Persamaan (3.1) dan Persamaan (3.2).

$$\text{Selisih Sudut} = \text{Sudut } \mathit{Output} - \text{Sudut Target} \tag{3.1}$$

$$Error = \frac{\text{Selisih Sudut}}{\text{Sudut Target}} \times 100\% \quad (3.2)$$

Dari Tabel 3.1 terlihat bahwa untuk mencapai target yang diinginkan, dibutuhkan nilai *i* minimum bernilai 11. Karena modifikasi perhitungan dilakukan secara paralel, maka *i* harus bernilai $2^n - 1$ agar sistem optimum, dengan n merupakan bilangan bulat positif. Sehingga sistem ini dilakukan iterasi hingga $i = 15$.

3. Karena iterasi dilakukan hingga $i = 15$, yang mana menghasilkan sudut ≈ 0,002˚, maka diperlukan 9 bit bilangan bulat, untuk *input* hingga 360˚, dan bilangan pecahan pada perhitungan diwakili dengan resolusi bilangan digital 9 bit. Sehingga total *input* yang digunakan pada sistem ini sebanyak 18 bit.
4. Jumlah bit pada sisi *output* diperoleh dari selisih terkecil nilai cos dan sin dengan resolusi sudut 0,002˚. Perhitungan selisih *output* dengan resolusi sudut 0,002˚ ditunjukkan pada Tabel 3.2.

Tabel 3.2 Perhitungan Selisih *Output* dengan Resolusi Sudut 0,002˚

Sudut (derajat)	Nilai Cos	Nilai Sin	Selisih Cos	Selisih Sin
0	1	0	-	-
0,002	0,9999999994	0,0000349065	6×10^{-10}	$3{,}49\times10^{-5}$
0,004	0,9999999976	0,0000698132	$1{,}8\times10^{-9}$	$3{,}49\times10^{-5}$
0,006	0,9999999945	0,0001047198	$3{,}1\times10^{-9}$	$3{,}49\times10^{-5}$
0,008	0,9999999903	0,0001396263	$4{,}2\times10^{-9}$	$3{,}49\times10^{-5}$
0,010	0,9999999848	0,0001745329	$5{,}5\times10^{-9}$	$3{,}49\times10^{-5}$
0,012	0,9999999781	0,0002094395	$6{,}7\times10^{-9}$	$3{,}49\times10^{-5}$

Dari Tabel 3.2 terlihat bahwa selisih *output* terkecil dapat mencapai 10^{-10}, sehingga diperlukan *output* bilangan pecahan dengan resolusi bilangan digital 30 bit. Nilai puncak cos atau sin adalah ±1, sehingga diperlukan 1 bit bilangan bulat dan 1 bit tanda. Total bit *output* pada sistem ini adalah 32 bit.

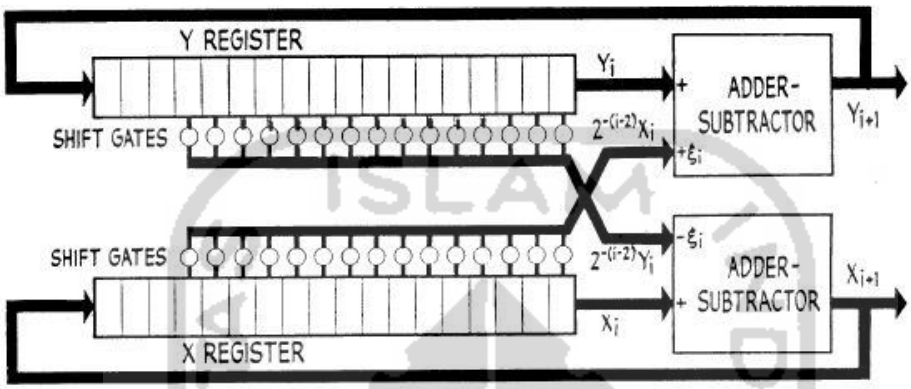
5. Sistem ini dirancang untuk *stepper* dengan kecepatan sampai dengan 3000 rpm atau sama dengan 50 rotasi per detik. *Stepper* motor dengan resolusi sudut 0,75˚/step, membutuhkan 480 step untuk melakukan satu kali rotasi. Sehingga *stepper* tersebut dapat melakukan step hingga 24000 step tiap detik. Maka sistem ini membutuhkan frekuensi minimum sebesar 24 kHz.
6. *Logic element* yang digunakan tidak lebih dari 80% untuk meminimalisir *delay* yang terjadi karena banyaknya jalur kritis yang digunakan.

3.4 Modifikasi Algoritme

Penelitian ini dilakukan dengan memodifikasi algoritme CORDIC dari perhitungan iterasi yang dilakukan secara serial menjadi paralelisasi perhitungan iterasi.

3.4.1 Algoritme CORDIC

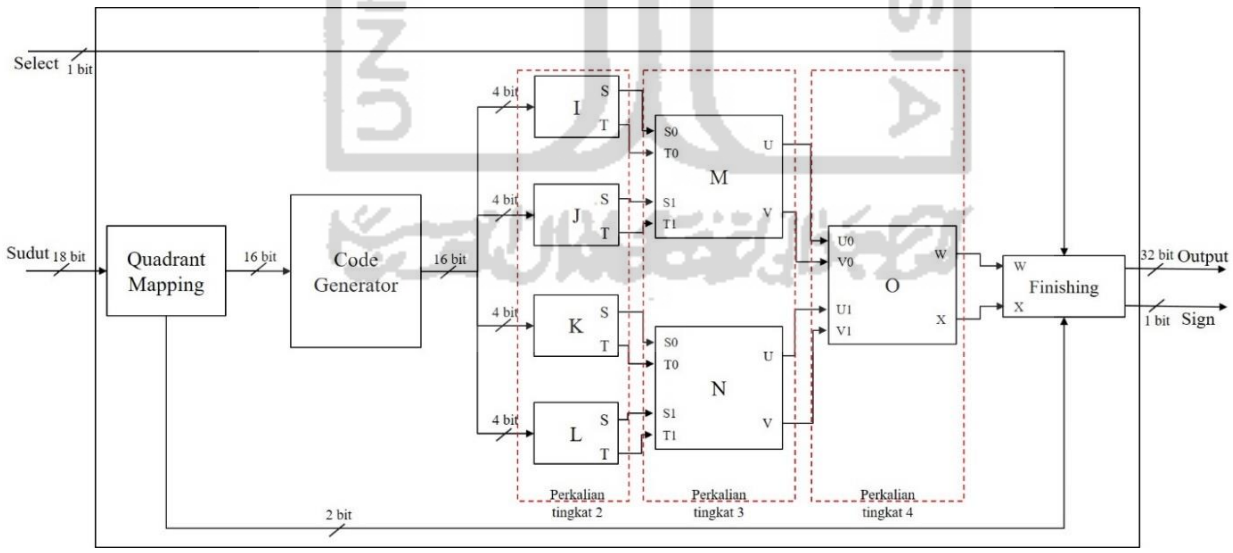
Secara umum, diagram blok algoritme CORDIC ditunjukkan oleh Gambar 3.2.



Gambar 3.2 Diagram Blok Algoritme CORDIC [2]

3.4.2 Hasil Modifikasi Algoritme CORDIC

Diagram blok sistem pada penelitian ini ditunjukkan oleh Gambar 3.3.



Gambar 3.3 Diagram Blok Sistem

a. *Quadrant Mapping*

Blok *Quadrant Mapping* berfungsi melakukan konversi sudut *input* menjadi sudut dalam kwadran 1 karena perhitungan cos dan sin suatu sudut di setiap kwadran dapat dinyatakan sebagai

nilai cos dan sin sudut pada kwadran 1. Blok ini mengubah *input* 18 bit menjadi 16 bit (7 bit bilangan bulat dan bilangan pecahan dengan resolusi bilangan digital 9 bit) dan 2 bit tanda hasil perhitungan trigonometri. Proses konversi dilakukan menggunakan Persamaan (3.3).

$$\theta_1 = \begin{cases} \theta_{in}, & 0° \le \theta_{in} \le 90° \\ 180° - \theta_{in}, & 90° \le \theta_{in} \le 180° \\ \theta_{in} - 180°, & 180° \le \theta_{in} \le 270° \\ 360° - \theta_{in}, & 270° \le \theta_{in} \le 180° \end{cases} \quad (3.3)$$

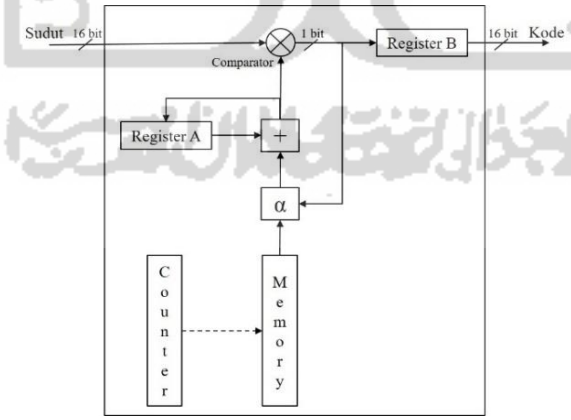
θ_1 = sudut kwadran 1

θ_{in} = sudut *input*

b. *Code Generator*

Blok ini menghasilkan 16 bit kode yang setiap bitnya merepresentasikan nilai α pada setiap iterasi. Saat sebuah bit kode bernilai 0 menandakan α bernilai +1, sebaliknya saat sebuah bit kode bernilai 1 menandakan α bernilai -1. Posisi setiap bit kode menyesuaikan dengan nomor iterasi, dengan *most significant bit* (MSB) sebagai nomor iterasi yang pertama dan *least significant bit* (LSB) sebagai nomor iterasi yang terakhir. Jika diketahui 4 bit kode yang berasal dari *code generator* adalah 0011, menunjukkan bahwa nilai α sebanyak 4 iterasi dengan arah rotasi positif, positif, negatif, dan negatif secara berturut-turut.

Proses yang terjadi pada *code generator* digambarkan melalui Gambar 3.4. Pada blok ini, sudut target yang merupkan *output* dari *quadrant mapping* dibandingkan dengan sudut *memory* algoritme CORDIC dan menghasilkan 1 bit kode. Sudut *memory* kemudian masuk ke register A untuk dijumlahkan dengan sudut *memory* berikutnya dan dibandingkan kembali dengan sudut target. Proses ini terjadi hingga 16 kali iterasi dan menghasilkan 16 bit kode.

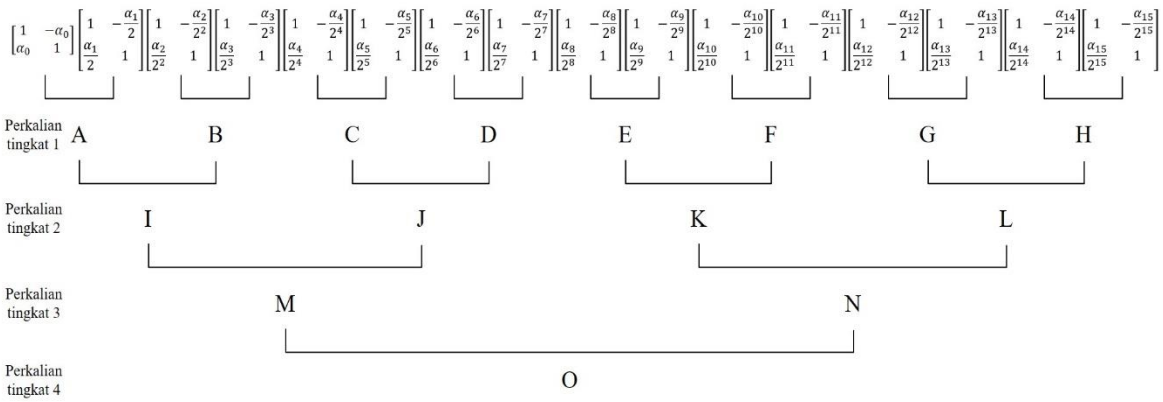


Gambar 3.4 Diagram Blok *Code Generator*

c. Blok Perkalian Matriks

Proses perkalian matriks pada sistem ini diilustrasikan oleh Gambar 3.5.

Perhitungan perkalian matriks tingkat 1, tingkat 2, tingkat 3, dan tingkat 4 dijabarkan pada Persamaan (3.4), Persamaan (3.5), Persamaan (3.6), dan Persamaan (3.7).



Gambar 3.5 Proses Paralelisasi Perkalian Matriks

$$\begin{bmatrix} 1 & -\frac{\alpha_i}{2^i} \\ \frac{\alpha_i}{2^i} & 1 \end{bmatrix}\begin{bmatrix} 1 & -\frac{\alpha_{i+1}}{2^{i+1}} \\ \frac{\alpha_{i+1}}{2^{i+1}} & 1 \end{bmatrix} = \begin{bmatrix} 1-\frac{\alpha_i\alpha_{i+1}}{2^{2i+1}} & -\frac{2\alpha_i+\alpha_{i+1}}{2^{i+1}} \\ \frac{2\alpha_i+\alpha_{i+1}}{2^{i+1}} & 1-\frac{\alpha_i\alpha_{i+1}}{2^{2i+1}} \end{bmatrix} = \begin{bmatrix} Q_{\frac{i}{2}} & -R_{\frac{i}{2}} \\ R_{\frac{i}{2}} & Q_{\frac{i}{2}} \end{bmatrix} \tag{3.4}$$

$$\begin{bmatrix} Q_j & -R_j \\ R_j & Q_j \end{bmatrix}\begin{bmatrix} Q_{j+1} & -R_{j+1} \\ R_{j+1} & Q_{j+1} \end{bmatrix} = \begin{bmatrix} Q_jQ_{j+1}-R_jR_{j+1} & -Q_jR_{j+1}-R_jQ_{j+1} \\ Q_jR_{j+1}+R_jQ_{j+1} & Q_jQ_{j+1}-R_jR_{j+1} \end{bmatrix} = \begin{bmatrix} S_{\frac{j}{2}} & -T_{\frac{j}{2}} \\ T_{\frac{j}{2}} & S_{\frac{j}{2}} \end{bmatrix} \tag{3.5}$$

$$\begin{bmatrix} S_k & -T_k \\ T_k & S_k \end{bmatrix}\begin{bmatrix} S_{k+1} & -T_{k+1} \\ T_{k+1} & S_{k+1} \end{bmatrix} = \begin{bmatrix} S_kS_{k+1}-T_kT_{k+1} & -S_kT_{k+1}-T_kS_{k+1} \\ S_kT_{k+1}+T_kS_{k+1} & S_kS_{k+1}-T_kT_{k+1} \end{bmatrix} = \begin{bmatrix} U_{\frac{k}{2}} & -V_{\frac{k}{2}} \\ V_{\frac{k}{2}} & U_{\frac{k}{2}} \end{bmatrix} \tag{3.6}$$

$$\begin{bmatrix} U_0 & -V_0 \\ V_0 & U_0 \end{bmatrix}\begin{bmatrix} U_1 & -V_1 \\ V_1 & U_1 \end{bmatrix} = \begin{bmatrix} U_0U_1-V_0V_1 & -U_0V_1-V_0U_1 \\ U_0V_1+V_0U_1 & U_0U_1-V_0V_1 \end{bmatrix} = \begin{bmatrix} W & -X \\ X & W \end{bmatrix} \tag{3.7}$$

Karena hasil perkalian matriks hanya bergantung pada kombinasi kode, sistem ini dapat langsung menghitung *output* W dan X tanpa perkalian bertingkat. Namun karena jumlah iterasi yang digunakan sebanyak 16 iterasi yang menghasilkan 16 bit kode, maka diperlukan 2^{16} kombinasi program untuk dapat memperoleh hasil dari semua kemungkinan kode. Oleh karena itu, sistem ini dirancang melalui perkalian matriks tingkat 2 yang hanya membutuhkan 2^4 kombinasi program tiap blok dan dua siklus perkalian matriks.

Perkalian matriks tingkat 2 ini membagi 16 bit kode ke dalam 4 blok I, J, K, dan L sehingga masing-masing blok memiliki 4 bit kode *input*. Proses perkalian pada perkalian tingkat 2 ini menggunakan *decoder* 4-bit untuk memperoleh nilai S dan T. Rincian kode dan *output* pada masing-masing blok dijabarkan pada Tabel 3.3.

Perkalian matriks tingkat 3 dan tingkat 4 dilakukan menggunakan modul multiplier 9-bit untuk menghemat banyaknya *logic element* yang digunakan. Untuk perkalian 32×32 bit membutuhkan 8 buah *multiplier* 9-bit dan menghasilkan *output* 64 bit. Sistem ini dirancang untuk menghasilkan *output* 32 bit, sedangkan 8 buah *multiplier* 9-bit dapat digunakan untuk perkalian 36×36 bit. Artinya, banyak *resource* yang terbuang sia-sia jika menggunakan perkalian 32×32 bit. Sehingga, perkalian matriks tingkat 3 dan tingkat 4 dilakukan dengan *input* 27×27 bit dan menghasilkan *output* 54 bit yang kemudian akan dipangkas menjadi 32 bit. Perkalian 27×27 bit

membutuhkan 7 buah *multiplier* 9-bit. Proses yang terjadi pada blok perkalian matriks dengan *multiplier* dirumuskan oleh Persamaan (3.8) dan Persamaan (3.9).

$$\text{out1} = \text{in11} \times \text{in21} - \text{in12} \times \text{in22} \quad (3.8)$$

$$\text{out2} = \text{in11} \times \text{in22} + \text{in12} \times \text{in21} \quad (3.9)$$

in11 = *output* pertama dari blok pertama perkalian matriks tingkat sebelumnya

in12 = *output* kedua dari blok pertama perkalian matriks tingkat sebelumnya

in21 = *output* pertama dari blok kedua perkalian matriks tingkat sebelumnya

in22 = *output* kedua dari blok kedua perkalian matriks tingkat sebelumnya

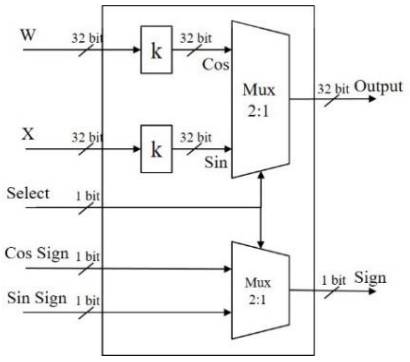
out1 = *output* pertama perkalian matriks

out2 = *output* kedua perkalian matriks

Dari Persamaan (3.8) dan Persamaan (3.9) diketahui bahwa perkalian matriks algoritme CORDIC menghasilkan dua buah *output* dengan dua buah perkalian pada masing-masing *output*. Maka satu buah blok perkalian matriks terdapat 4 buah perkalian dan membutuhkan 28 buah *multiplier* 9-bit. Sehingga untuk tiga blok perkalian matriks tingkat 3 dan tingkat 4 membutuhkan 84 *multiplier* 9-bit. Sedangkan FPGA Cyclone IV EP4CE6E22C8N hanya memiliki 30 *multiplier* 9-bit. Untuk mengurangi *multiplier* yang digunakan, maka proses perkalian dilakukan secara bergantian untuk setiap bloknya. Sehingga setiap blok hanya membutuhkan 7 buah *multiplier* 9-bit dan total *multiplier* 9-bit yang dibutuhkan hanya berjumlah 21 buah. Hal ini menyebabkan proses komputasi menjadi lebih lama karena menambah 4 siklus *clock* pada setiap blok yang menggunakan modul *multiplier*.

d. *Finishing*

Blok *Finishing* merupakan proses terakhir dari sistem ini. Blok ini berisi perkalian konstanta dengan hasil perkalian paralel matriks. Blok ini juga berisi perintah untuk memilih *output* yang akan ditampilkan berdasarkan *input select*. Proses yang terjadi pada blok ini digambarkan melalui Gambar 3.6.



Gambar 3.6 Diagram Blok *Finishing*

Tabel 3.3 Penjabaran Kode Perkalian Matriks Tingkat 2

Blok	Nomor Bit Kode	Bit Kode	S	T
I	15 – 12	0000	-0,078125	1,640625
		0001	0,328125	1,609375
		0010	0,703125	1,484375
		0011	1,046875	1,265625
		0100	1,265625	1,046875
		0101	1,484375	0,703125
		0110	1,609375	0,328125
		0111	1,640625	-0,078125
		1000	1,640625	0,078125
		1001	1,609375	-0,328125
		1010	1,484375	-0,703125
		1011	1,265625	-1,046875
		1100	1,046875	-1,265625
		1101	0,703125	-1,484375
		1110	0,328125	-1,609375
		1111	-0,078125	-1,640625
J	11 – 8	0000	0,995727777	0,11713028
		0001	0,997436285	0,101558685
		0010	0,998901129	0,085964203
		0011	1,000122309	0,070346832
		0100	1,001098394	0,054729462
		0101	1,001831293	0,039081573
		0110	1,002319574	0,023426056
		0111	1,002563238	0,007762909
		1000	1,002563238	-0,007762909
		1001	1,002319574	-0,023426056
		1010	1,001831293	-0,039081573
		1011	1,001098394	-0,054729462
		1100	1,000122309	-0,070346832
		1101	0,998901129	-0,085964203
		1110	0,997436285	-0,101558685
		1111	0,995727777	-0,11713028
K	7 – 4	0000	0,999983311	0,007324205
		0001	0,999989986	0,006347655
		0010	0,999995708	0,0053711
		0011	1,000000477	0,00439454
		0100	1,000004292	0,003417979
		0101	1,000007153	0,002441411
		0110	1,00000906	0,001464841
		0111	1,000010014	0,000488269
		1000	1,000010014	-0,000488269
		1001	1,00000906	-0,001464841
		1010	1,000007153	-0,002441411
		1011	1,000004292	-0,003417979
		1100	1,000000477	-0,00439454
		1101	0,999995708	-0,0053711
		1110	0,999989986	-0,006347655
		1111	0,999983311	-0,007324205
L	3 – 0	0000	0,999999935	0,000457764
		0001	0,999999961	0,000396729
		0010	0,999999983	0,000335693
		0011	1,000000002	0,000274658
		0100	1,000000017	0,000213623
		0101	1,000000028	0,000152588
		0110	1,000000035	0,000091553
		0111	1,000000039	0,000030518
		1000	1,000000039	-0,000030518
		1001	1,000000035	-0,000091553
		1010	1,000000028	-0,000152588
		1011	1,000000017	-0,000213623
		1100	1,000000002	-0,000274658
		1101	0,999999983	-0,000335693
		1110	0,999999961	-0,000396729
		1111	0,999999935	-0,000457764

Pemilihan nilai cos atau sin sebagai *output* menggunakan *multiplexer* 2 ke 1. *Output* akan menampilkan nilai cos saat *select* bernilai 1, dan akan menampilkan nilai sin saat *select* bernilai 0.

3.5 Verifikasi Sistem

Proses verifikasi sistem dilakukan dalam dua tahap, yaitu verifikasi fungsional dengan aplikasi ModelSim-Altera dan verifikasi *hardware* dengan modul FPGA Cyclone IV EP4CE6E22C8N.

Verifikasi fungsional dilakukan dengan menjalankan simulasi menggunakan beberapa *input* berbeda kemudian membandingkan hasilnya dengan nilai *real*. Pada verifikasi fungsional, semua bilangan dinyatakan dalam bilangan bulat, sehingga perlu dilakukan konversi bilangan untuk merepresentasikan bilangan pecahan. Konversi bilangan bulat ke bilangan pecahan dirumuskan melalui Persamaan (3.10).

$$x_f = \frac{x_m}{2^r} \tag{3.10}$$

x_f = bilangan pecahan

x_m = bilangan bulat

r = jumlah bit di belakang koma

Pada sistem ini hanya terdapat dua nilai *r*, yaitu 9 bit di sisi *input* dan 30 bit di sisi *output*.

Pada verifikasi *hardware*, dilakukan penambahan program untuk menyesuaikan dengan *port* I/O *board* FPGA. Verifikasi *hardware* ini menggunakan *port* reset, *clock*, dan 3 buah *push button* sebagai *input*, serta 4 *port seven segment* dan satu *port* LED sebagai penampil nilai *input* dan *output*. *Input* sudut diperoleh dari *push button* sebagai *counter up/down* yang kemudian dikonversi ke dalam bilangan biner 18 bit. Karena keterbatasan penampil, proses verifikasi *hardware* hanya dapat dilakukan untuk 4 digit bilangan desimal. *Seven segment* menampilkan nilai *input* dan *output* secara bergantian sesuai isyarat dari *push button*. LED mengindikasikan bahwa proses telah selesai dilaksanakan.

3.6 Mekanisme Pengujian

Mekanisme pengujian berisi langkah-langkah yang dilakukan untuk menilai kinerja sistem yang telah dibuat. Langkah pertama adalah melakukan pengujian fungsional pada setiap blok. Pemrograman VHDL pada setiap blok dibuat secara terpisah kemudian dilakukan simulasi pada masing-masing program. Pengujian fungsional setiap blok bertujuan untuk memastikan kinerja sistem dari masing-masing blok menghasilkan hubungan *input-output* yang sesuai dengan hasil modifikasi algoritme CORDIC.

Output pada blok *Quadrant Mapping* harus sama dengan nilai sudut kwadran I dari sudut *input*. Pada blok *Code Generator*, *output* yang dihasilkan adalah 16 bit kode yang dapat merepresentasikan sudut kwadran I dengan toleransi 5%. Blok-blok perkalian matriks tingkat 2 harus menghasilkan hubungan *input-output* sesuai dengan Tabel 3.3, sedangkan blok-blok perkalian matriks tingkat 3 dan 4 melakukan perhitungan menggunakan Persamaan (3.8) dan Persamaan (3.9). Blok *Finishing* melakukan proses perkalian dan pemilihan *output* seperti pada Gambar 3.6.

Langkah kedua adalah melakukan verifikasi fungsional dengan menggabungkan seluruh program pada masing-masing blok ke dalam suatu program utama.menggunakan *component statement*. Kemudian hasil verifikasi fungsional dibandingkan dengan nilai *real* untuk mengetahui seberapa besar perbedaan nilai yang dihasilkan. Suatu sistem dikatakan memiliki kinerja yang baik apabila memiliki *error* tidak lebih dari 5%.

Pada verifikasi fungsional akan terlihat *resources* yang terlibat dalam sistem ini. *Resources* ini akan dibandingkan dengan penelitian lain yang sejenis untuk mengetahui pengaruh paralelisasi komputasi pada algoritme CORDIC terhadap *latency* yang dihasilkan.

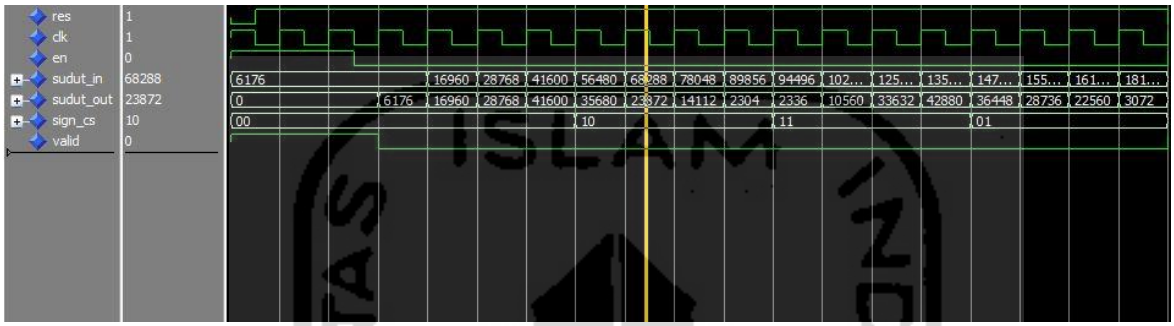
Langkah terakhir adalah melakukan verifikasi *hardware*. Proses pengujian pada verifikasi *hardware* dilakukan dengan membuat konfigurasi *port* sesuai dengan I/O yang dibutuhkan sehingga dapat menampilkan nilai *input* dan *output* sesuai dengan nilai *real*.

BAB 4
HASIL DAN PEMBAHASAN

4.1 Pengujian Fungsional Setiap Blok

4.1.1 *Quadrant Mapping*

Hasil dari pengujian *Quadrant Mapping* ditunjukkan oleh Gambar 4.1.



Gambar 4.1 Hasil Pengujian *Quadrant Mapping*

Nilai-nilai pada Gambar 4.1 dijabarkan melalui Tabel 4.1.

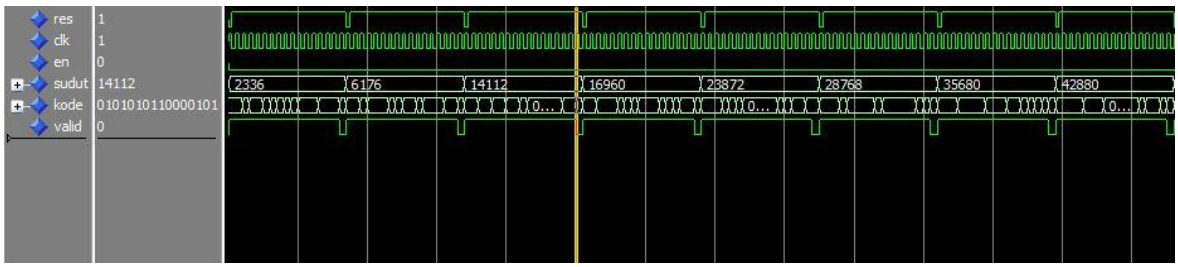
Tabel 4.1 Penjabaran Hasil Pengujian *Quadrant Mapping*

Nilai *Input* (desimal)	Sudut *Input* (derajat)	Kwadran	Sudut Kwadran I (derajat)	Tanda Cos		Tanda Sin		Nilai *Output* (desimal)	Sudut *Output* (derajat)
6176	12,0625	I	12,0625	Positif	0	Positif	0	6176	12,0625
16960	33,1250		33,1250					16960	33,1250
28768	56,1875		56,1875					28768	56,1875
41600	81,2500		81,2500					41600	81,2500
55480	110,3125	II	69,6875	Negatif	1	Positif	0	35680	69,6875
68288	133,3750		46,6250					23872	46,6250
78048	152,4375		27,5625					14112	27,5625
89856	175,5000		4,5000					2304	4,5000
94496	184,5625	III	4,5625	Negatif	1	Negatif	1	2336	4,5625
102720	200,6250		20,6250					10560	20,6250
125792	245,6875		65,6875					33632	65,6875
135040	263,7500		83,7500					42880	83,7500
147872	288,8125	IV	71,1875	Positif	0	Negatif	1	36448	71,1875
155584	303,8750		56,1250					28736	56,1250
161760	315,9375		44,0625					22560	44,0625
181248	354		6					3072	6

Dari Tabel 4.1 terlihat bahwa nilai sudut *output* telah menghasilkan nilai yang sama dengan sudut kwadran I. Selain itu, blok *Quadrant Mapping* juga telah menghasilkan 2 bit tanda yang menunjukkan kwadran asal dari sudut *input*.

4.1.2 *Code Generator*

Gambar 4.2 menunjukkan hasil pengujian pada blok *Code Generator*.



Gambar 4.2 Hasil Pengujian *Code Generator*

Nilai-nilai pada Gambar 4.2 dijabarkan melalui Tabel 4.2.

Tabel 4.2 Penjabaran Hasil Pengujian *Code Generator*

Nilai *Input* (desimal)	Sudut *Input* (derajat)	Kode α	Hasil Penjumlahan Sudut CORDIC	Selisih Sudut (derajat)	*Error* (%)
2336	4,5625	0110111111001001	4,5576	0,0049	0,1075
6176	12,0625	0110011101100101	12,0600	0,0025	0,0207
14112	27,5625	0101010110000101	27,5609	0,0016	0,0058
16960	33,1250	0100111100111101	33,1197	0,0053	0,0160
23872	46,6250	0011110000111010	46,6234	0,0016	0,0034
28768	56,1875	0011000110000111	56,1914	0,0039	0,0069
35680	69,6875	0010001001011111	69,6862	0,0013	0,0019
42880	83,7500	0001001000011011	83,7464	0,0036	0,0043

Hasil penjumlahan sudut CORDIC merupakan representasi kode α berdasarkan sudut *input*. Nilai ini didapat dari menjumlahkan sudut-sudut dari Tabel 2.1 dengan arah rotasi α. Dari Tabel 4.2 terlihat bahwa kode *α* telah merepresentasikan nilai *input* dengan *error* tidak lebih dari 5%.

4.1.3 Perkalian Matriks

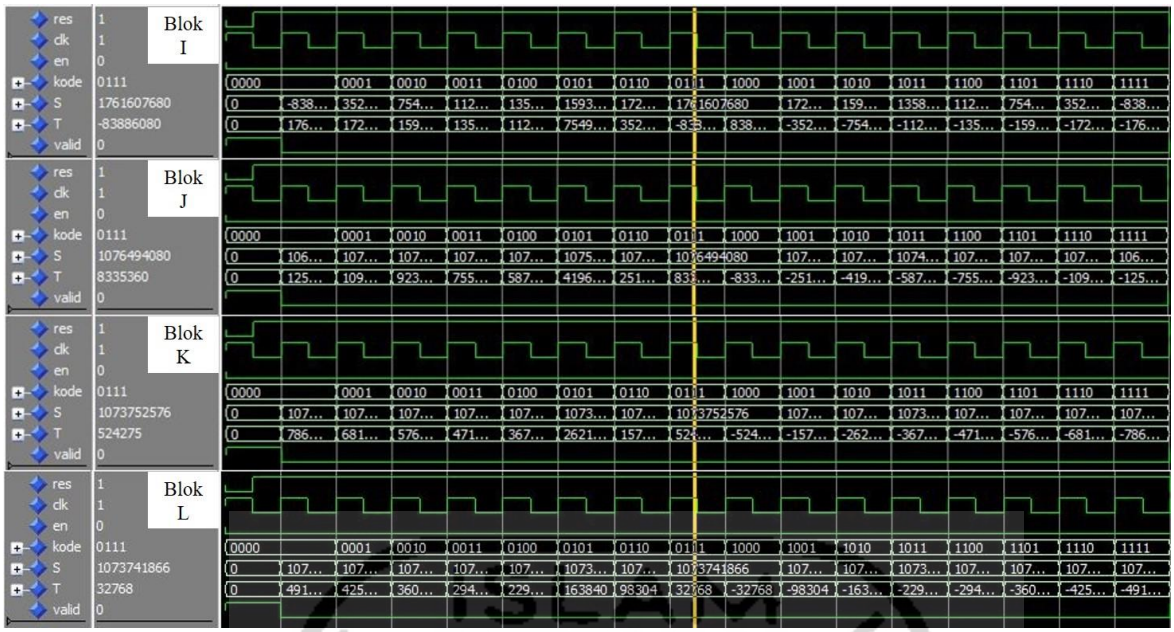
Hasil pengujian perkalian matriks tingkat 2 menggunakan *decoder* 4-bit ditunjukkan oleh Gambar 4.3.

Output S dan T pada Gambar 4.3 merupakan bentuk desimal 32 bit yang merepresentasikan nilai S dan T pada Tabel 3.3.

Hasil pengujian perkalian matriks tingkat 3 dan 4 menggunakan modul *multiplier* 9-bit ditunjukkan oleh Gambar 4.4.

Nilai-nilai pada Gambar 4.4 dijabarkan melalui Tabel 4.3.

Dari Tabel 4.3 terlihat bahwa hasil pengujian blok perkalian matriks tingkat 3 dan 4 menghasilkan nilai yang sesuai perhitungan menggunakan Persamaan (3.8) dan Persamaan (3.9).



Gambar 4.3 Hasil Pengujian Perkalian Matriks Tingkat 2 dengan *Decoder* 4-Bit

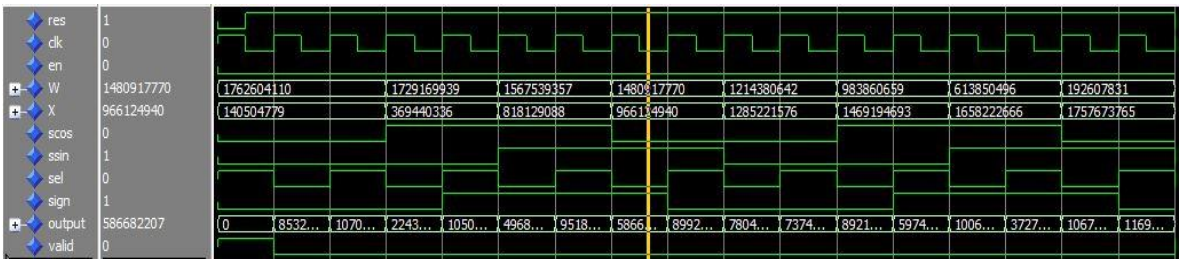
Gambar 4.4 Hasil Pengujian Perkalian Matriks dengan Modul *Multiplier* 9-Bit

Tabel 4.3 Penjabaran Hasil Pengujian Perkalian Matriks dengan Modul *Multiplier* 9-Bit

	Input				*Output*		Hasil Perhitungan	
	in11	in12	in21	in22	out1	out2	out1	out2
desimal	1728053248	352321536	1069154560	-125767680	1761938140	148408980	1,64093277	0,13821663
pecahan	1,60937500	0,32812500	0,99572778	-0,11713028	1,64093276	0,13821663		
desimal	1073742336	-4718601	1073741862	-98304	1073741942	-4816905	1,00000012	-0,00448605
pecahan	1,00000048	-0,00439450	1,00000004	-0,00009155	1,00000011	-0,00448609		
desimal	1593835520	754974720	1075708160	41963520	1567248700	818646900	1,45961409	0,76242433
pecahan	1,48437500	0,70312500	1,00183129	0,03908157	1,45961409	0,76242434		
desimal	1073752576	-524275	1073741854	163840	1073752685	-360434	1,00001012	-0,00033571
pecahan	1,00001001	-0,00048830	1,00000003	0,00015259	1,00001012	-0,00033568		
desimal	1124073472	1358954496	1073873152	-75534336	1219809100	1280045700	1,13603575	1,19213546
pecahan	1,04687500	1,26562500	1,00012231	-0,07034683	1,13603575	1,19213546		
desimal	1073742336	4718601	1073741854	-163840	1073743086	4554761	1,00000118	0,00424195
pecahan	1,00000048	0,00439454	1,00000003	-0,00015259	1,00000118	0,00424195		
desimal	754974720	1593835520	1072561920	92303360	617132300	1656984900	0,57474925	1,54318744
pecahan	0,70312500	1,48437500	0,99890113	0,08596420	0,57474924	1,54318744		
desimal	1073749504	2621445	1073741754	-491520	1073750633	2129916	1,00000820	0,00198365
pecahan	1,00000715	0,00244141	0,99999993	-0,00045776	1,00000820	0,00198364		

4.1.4 *Finishing*

Hasil pengujian blok *finishing* ditunjukkan oleh Gambar 4.5.



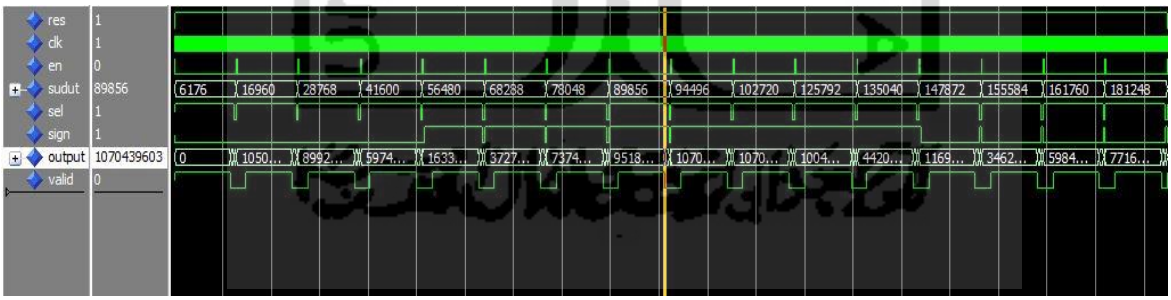
Gambar 4.5 Hasil Pengujian Blok *Finishing*

Nilai-nilai pada Gambar 4.5 dijabarkan melalui Tabel 4.4.

Dari Tabel 4.4 terlihat bahwa *output* yang dihasilkan blok *Finishing* telah sesuai dengan hasil perhitungan berdasarkan pemilihan *select*, dengan *output sign* sebagai penanda posistif atau negatif, *sign* bernilai 0 menandakan *output* bernilai positif dan *sign* bernilai 1 menandakan *output* bernilai negatif.

4.2 Verifikasi Fungsional

Proses verifikasi fungsional dilakukan menggunakan variasi sudut *input* sebanyak 16 buah di 4 kwadran berbeda untuk menghasilkan nilai cos dan sin dari masing-masing sudut. Hasil verifikasi fungsional sistem ditunjukkan oleh Gambar 4.6.



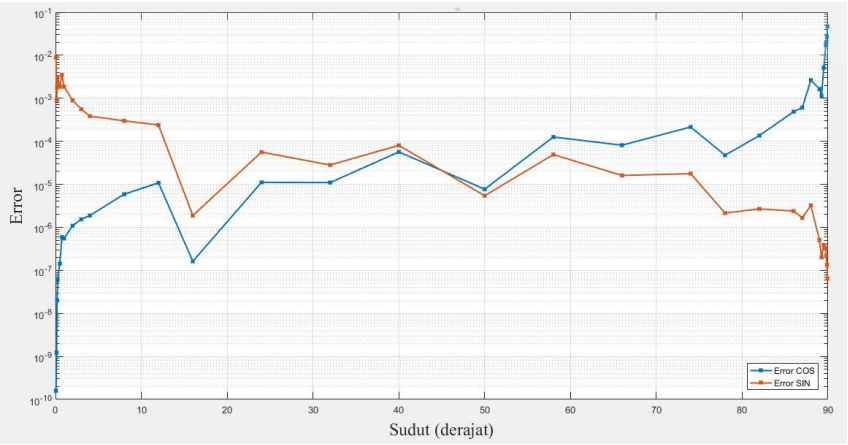
Gambar 4.6 Hasil Verifikasi Fungsional

Nilai-nilai pada Gambar 4.6 dijabarkan melalui Tabel 4.5.

Dari Tabel 4.5 terlihat bahwa nilai *output* sistem ini memberikan hasil yang cukup akurat dengan *error* yang sangat kecil. *Error* tersebut disebabkan oleh perancangan sistem yang hanya menggunakan 9 bit bilangan yang merepresentasikan pecahan pada blok *Code Generator* serta pemangkasan *input* blok perkalian dengan *multiplier* dari 32 bit menjadi 27 bit. Gambar 4.7 menunjukkan grafik hubungan sudut *input* dengan nilai *error* untuk sudut dengan resolusi 0,0625˚.

Tabel 4.4 Penjabaran Hasil Pengujian Blok *Finishing*

	W	X	Kwadran	*Select*	*Sign*		*Output*	Hasil Perhitungan
Desimal	1762604110	140504779	I	0	0	Desimal	85321941	0,07946224
						Pecahan	0,07946225	
Pecahan	1,64155300	0,13085527		1	0	Desimal	1070346520	0,99683782
						Pecahan	0,99683788	
Desimal	1729169939	369440336	II	0	0	Desimal	224343730	0,20893636
						Pecahan	0,20893638	
Pecahan	1,61041500	0,34406812		1	1	Desimal	1050043522	-0,97792918
						Pecahan	0,97792924	
Desimal	1567539357	818129088	III	0	1	Desimal	496811291	-0,46269154
						Pecahan	0,46269157	
Pecahan	1,45988479	0,76194209		1	1	Desimal	951892878	-0,88651927
						Pecahan	0,88651933	
Desimal	1480917770	966124940	IV	0	1	Desimal	586682207	-0,54639035
						Pecahan	0,54639038	
Pecahan	1,37921215	0,89977397		1	0	Desimal	899291665	0,83753058
						Pecahan	0,83753063	
Desimal	1214380642	1285221576	I	0	0	Desimal	780454576	0,72685492
						Pecahan	0,72685497	
Pecahan	1,13098011	1,19695587		1	0	Desimal	737436210	0,68679095
						Pecahan	0,68679099	
Desimal	983860659	1469194693	II	0	0	Desimal	892172791	0,83090061
						Pecahan	0,83090066	
Pecahan	0,91629164	1,36829418		1	1	Desimal	597452276	-0,55642076
						Pecahan	0,55642079	
Desimal	613850496	1658222666	III	0	1	Desimal	1006960582	-0,93780507
						Pecahan	0,93780512	
Pecahan	0,57169282	1,54434020		1	1	Desimal	372762517	-0,34716212
						Pecahan	0,34716215	
Desimal	192607831	1757673765	IV	0	1	Desimal	1067352553	-0,99404947
						Pecahan	0,99404953	
Pecahan	0,17938002	1,63696126		1	0	Desimal	116961672	0,10892904
						Pecahan	0,10892905	



Gambar 4.7 Grafik Hubungan Sudut *Input* dengan Persentase *Error*

Tabel 4.5 Penjabaran Hasil Verifikasi Fungsional

Sudut		*Select*	*Sign*	*Output*		Hasil Perhitungan		*Error* (%)
Desimal	Derajat			Desimal	Pecahan			
6176	12,0625	1	0	1050043522	0,97792924	Cos	0,97792022	0,00092237
		0	0	224343725	0,20893638	Sin	0,20897856	0,02018389
16960	33,1250	1	0	899291667	0,83753063	Cos	0,83748035	0,00600372
		0	0	586682200	0,54639038	Sin	0,54646743	0,01409965
28768	56,1875	1	0	597452275	0,55642079	Cos	0,55647689	0,01008128
		0	0	892172791	0,83090066	Sin	0,83086308	0,00452301
41600	81,2500	1	0	163331238	0,15211407	Cos	0,15212339	0,00612661
		0	0	1061246620	0,98836293	Sin	0,98836151	0,00014367
56480	110,3125	1	1	372762522	0,34716215	Cos	-0,34714026	0,00630581
		0	0	1006960577	0,93780512	Sin	0,93781322	0,00086371
68288	133,3750	1	1	737436209	0,68679099	Cos	-0,68677042	0,00299518
		0	0	780454574	0,72685496	Sin	0,72687440	0,00267446
78048	152,4375	1	1	951892876	0,88651932	Cos	-0,88650662	0,00143259
		0	0	496811288	0,46269157	Sin	0,46271592	0,00526241
89856	175,5000	1	1	1070439603	0,99692457	Cos	-0,99691733	0,00072624
		0	0	84145970	0,07836704	Sin	0,07845910	0,11733502
94496	184,5625	1	1	1070346517	0,99683787	Cos	-0,99683116	0,00067313
		0	1	85321941	0,07946225	Sin	-0,07954652	0,10593801
102720	200,6250	1	1	1004967015	0,93594847	Cos	-0,93590593	0,00454533
		0	1	378104216	0,35213699	Sin	-0,35225005	0,03209652
125792	245,6875	1	1	442087376	0,41172595	Cos	-0,41171319	0,00309924
		0	1	978509192	0,91130770	Sin	-0,91131348	0,00063425
135040	263,7500	1	1	116961669	0,10892904	Cos	-0,10886687	0,05710645
		0	1	1067352544	0,99404952	Sin	-0,99405634	0,00068608
147872	288,8125	1	0	346211920	0,32243498	Cos	0,32247221	0,01154518
		0	1	1016395008	0,94659162	Sin	-0,94657893	0,00134062
155584	303,8750	1	0	598486493	0,55738398	Cos	0,55738289	0,00019556
		0	1	891479347	0,83025484	Sin	-0,83025557	0,00008793
161760	315,9375	1	0	771622678	0,71862962	Cos	0,71858162	0,00667983
		0	1	746672576	0,69539303	Sin	-0,69544264	0,00713359
181248	354	1	0	1067862837	0,99452477	Cos	0,99452190	0,00028858
		0	1	112207184	0,10450108	Sin	-0,10452846	0,02619382

Dari Gambar 4.7 terlihat bahwa penggunaan sudut *input* dengan resolusi 0,0625˚ menghasilkan *error* perhitungan kurang dari 5%. Artinya sistem ini dapat digunakan untuk menghitung cos atau sin dari sudut dengan resolusi hingga 0,0625˚.

Total *resource* yang digunakan pada sistem ini ditunjukkan oleh Tabel 4.6.

Untuk mendapatkan nilai *output*, sistem ini membutuhkan 29 siklus *clock* dengan frekuensi maksimum 48,65 MHz. Sehingga waktu *latency* yang dibutuhkan sekitar 596,1 ns untuk melakukan satu kali proses dari aktivasi dengan *enable* hingga didapat nilai *output*.

Tabel 4.6 Total *Resource* yang Digunakan

Resource	Jumlah
Logic Element	3336 / 6272 (53%)
Register	815
Memory Bit	0 / 276480 (0%)
Embedded Multiplier 9-Bit	21 / 30 (70%)

4.3 Verifikasi *Hardware*

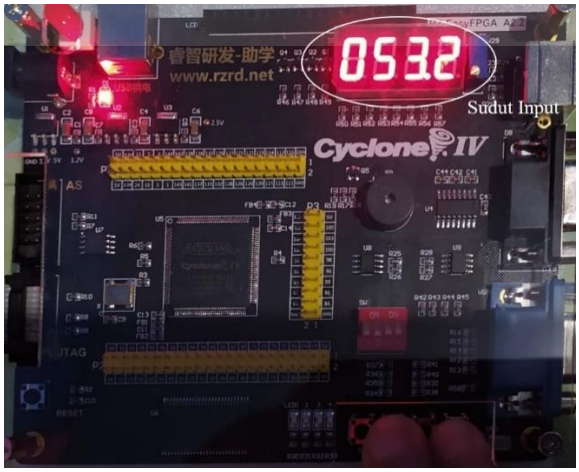
Proses verifikasi *hardware* melibatkan *port* reset, *clock*, dan 3 buah *port push button* sebagai *input*, serta 4 *port seven segment* dan satu *port* LED sebagai *output*. Konfigurasi isyarat *push button* ditunjukkan oleh Tabel 4.7.

Tabel 4.7 Konfigurasi Isyarat *Push Button*

S4	S3	S2	Penampil *Seven Segment*
On	*Counter down*	*Counter up*	*Input*
Off	*Enable*	*Select*	*Output*

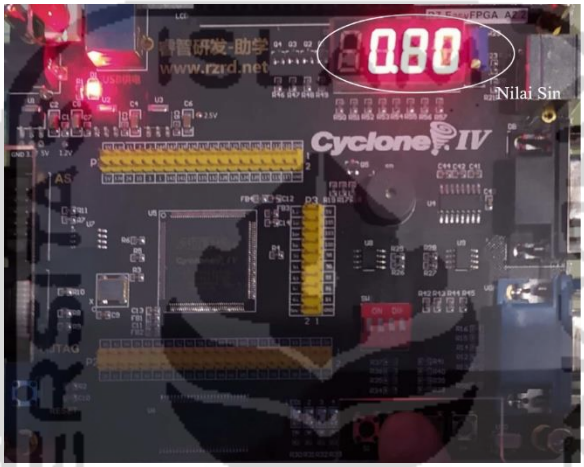
Karena keterbatasan penampil *seven segment*, maka proses verifikasi *hardware* hanya dapat dilakukan dengan resolusi sudut 0,1˚ dan resolusi *output* hingga 0,01. Proses verifikasi *hardware* ditunjukkan oleh Gambar 4.8 hingga Gambar 4.10.

Karena hanya terdapat empat buah *seven segment*, maka proses verifikasi *hardware* hanya dapat menampilkan *input* hingga resolusi 0,1˚ dan *output* hingga resolusi 0,01 atau hanya menggunakan 7 bit bilangan yang merepresentasikan nilai pecahan pada sisi *output*. Sehingga terdapat beberapa *output* yang tidak mengalami pembulatan nilai karena pemangkasan bit tersebut. Akibatnya, pada verifikasi *hardware* terdapat beberapa selisih *output* hingga 0,01.



Gambar 4.8 Verifikasi *Hardware*: Memilih Sudut *Input*

Gambar 4.9 Verifikasi *Hardware*: Aktivasi Proses



Gambar 4.10 Verifikasi *Hardware*: Memilih *Output*

4.4 Perbandingan dengan Hasil Penelitian Sejenis

Hasil penelitian yang dilibatkan dalam perbandingan ini adalah penelitian [12]–[14]. Penelitian-penelitian ini melakukan komputasi pada modifikasi algoritme CORDIC hingga 16 kali iterasi, namun dengan resolusi bit *output* yang jauh lebih kecil, yaitu 16 bit.

Hasil perbandingan dijelaskan dalam Tabel 4.8.

Tabel 4.8 Perbandingan dengan Hasil Penelitian Lain

Aspek	Valls [12]	Juang [13]	Maharatna [14]	Penelitian Ini
Iterasi	16	16	12	16
LUT	1606	3792	3072	3199
Register	1071	NA	597	815
Kecepatan *Clock*	25 MHz	54 MHz	20 MHz	48,65 MHz
Multiplier	NA	NA	NA	21
DSP	NA	NA	NA	0
Latency	680 ns	482 ns	700 ns	596,1 ns

Dari Tabel 4.8 terlihat bahwa penelitian ini memiliki keunggulan pada aspek frekuensi dan *latency* dibandingkan dengan penelitian [12] dan [14] yang melakukan komputasi secara serial, serta keunggulan pada aspek LUT dibandingkan dengan penelitian [13] yang juga menggunakan teknik paralelisasi. Penelitian ini menggunakan resolusi *output* hingga 30 bit sehingga sistem ini memiliki akurasi yang lebih tinggi dibandingkan dengan penelitian lain, namun menyebabkan jumlah LUT dan waktu *latency* menjadi lebih tinggi. Selain itu, tingginya jumlah LUT dan waktu *latency* juga disebabkan oleh perancangan sistem ini yang masih membutuhkan proses iterasi secara serial untuk mendapatkan arah rotasi sudut CORDIC.

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan dari penelitian ini adalah sebagai berikut:

1. Proses paralelisasi algoritme CORDIC dirancang untuk melakukan perkalian matriks algoritme CORDIC secara bersamaan. Pada penelitian ini, perkalian matriks dirancang dalam dua jenis, yaitu perkalian menggunakan *decoder* 4-bit dan perkalian dengan blok *multiplier* 9-bit untuk menghemat jumlah *logic element* yang digunakan.
2. Proses verifikasi sistem dilakukan dalam dua tahap, yaitu verifikasi fungsional dilakukan dengan mensimulasikan program dengan aplikasi ModelSim-Altera menggunakan beberapa nilai *input* yang berbeda kemudian membandingkan hasilnya dengan nilai *real*; serta verifikasi *hardware* menggunakan perangkat FPGA dengan menyesuaikan *input* dan *output* sistem dengan I/O *board* FPGA.
3. Sistem ini menggunakan 3336 *logic elements* dan 21 *multiplier* 9-bit, serta menghasilkan 29 siklus *clock* dengan waktu *latency* 596,1 ns untuk melakukan satu kali proses.
4. Teknik paralelisasi algoritme CORDIC dapat mempercepat waktu *latency* yang dibutuhkan, namun akan meningkatkan jumlah LUT yang digunakan. Selain teknik paralelisasi, resolusi bit *input* dan *output* juga dapat mempengaruhi waktu *latency*, semakin besar resolusi bit *input* dan *output* akan menyebabkan waktu *latency* menjadi lebih lama.

5.2 Saran

1. Waktu *latency* dapat dipercepat dengan memperkecil resolusi bit *input* atau *output*, namun akan memperkecil akurasi yang dihasilkan.
2. Untuk penggunaan resolusi sudut yang lebih tinggi, dapat dilakukan dengan menambahkan jumlah iterasi dan memperlebar jumlah bit yang digunakan, namun akan memperlambat waktu *latency* yang dibutuhkan.
3. Perhitungan trigonometri lain (*tangent*, *secant*, *cosecant*, dan *cotangent*) dapat diperoleh dari penurunan nilai cos dan sin.

DAFTAR PUSTAKA

[1] T. Sasao, J. Butler, and M. Riedel, "Application of LUT Cascades to Numerical Function Generators," in *SASIMI 2004 Proceedings (4-7)*, 2004, pp. 422–429.

[2] J. Volder, "The CORDIC Computing Technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, 1959.

[3] Nippon Pulse Motor Co., "Stepper Motors," 2015.

[4] A. Saha, A. Ghosh, and K. G. Kumar, "FPGA Implementation of Arcsine Function Using CORDIC Algorithm," *Adv. Model. Anal. A*, vol. 54, no. 2, pp. 197–202, 2017.

[5] F. de Dinechin, M. Istoan, and G. Sergent, "Fixed-Point Trigonometric Functions on FPGAs," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 5, pp. 83–88, 2014.

[6] D. Zhang, J. Yu, and Y. Song, "Implementation of High Accuracy Trigonometric Function on FPGA by Taylor Expansion," in *International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2016)*, 2016, pp. 175–179.

[7] T. Lang and M. D. Ercegovac, "CORDIC Algorithm and Implementation," in *Digital Arithmetic*, Morgan Kaufmann Publishers, 2004.

[8] R. Andraka, "A Survey of CORDIC Algorithms for FPGA Based Computers," in *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, 1998, pp. 191–200.

[9] C. Unsalan and B. Tar, "Field Programmable Gate Arrays," in *Digital System Design with FPGA Implementation Using Verilog and VHDL*, McGraw-Hill Education, 2017, pp. 5–16.

[10] Altera Coorporation, "Cyclone IV FPGA Device Family Overview," 2016.

[11] Teknik Elektro Universitas Islam Indonesia, *Buku Petunjuk Praktikum Sistem Digital*, 4th ed. Yogyakarta: Teknik Elektro Universitas Islam Indonesia, 2019.

[12] J. Valls, M. Kuhlmann, and K. K. Parhi, "Evaluation of CORDIC Algorithms for FPGA Design," *J. VLSI Signal Process. Syst. Signal Image. Video Technol.*, vol. 32, no. 3, pp. 207–222, 2002.

[13] T. Juang, S. Hsiao, and M. Tsai, "Para-CORDIC : Parallel CORDIC Rotation Algorithm," *IEEE Trans. Circuits Syst.*, vol. 51, no. 8, pp. 1515–1524, 2004.

[14] K. Maharatna, S. Banerjee, E. Grass, M. Krstic, and A. Troya, "Modified Virtually Scaling-Free Adaptive CORDIC Rotator Algorithm and Architecture," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 11, pp. 1463–1474, 2005.