TUGAS AKHIR-EE 184810

LOKALISASI SUARA MENGGUNAKAN FPGA BERBASIS KOMPUTASI STOKASTIK

Rifqi Yunus Trisna Pratama
NRP 07111540000108

Dosen Pembimbing
Astria Nur Irfansyah, ST., M.Eng., Ph.D
Fajar Budiman, S.T., M.Eng.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR-EE 184810

LOKALISASI SUARA MENGGUNAKAN FPGA BERBASIS KOMPUTASI STOKASTIK

Rifqi Yunus Trisna Pratama
NRP 07111540000108

Dosen Pembimbing
Astria Nur Irfansyah, ST., M.Eng., Ph.D
Fajar Budiman, S.T., M.Eng.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

FINAL PROJECT–EE 184801

SOUND LOCALIZATION USING FPGA BASED ON STOCHASTIC COMPUTATION

Rifqi Yunus Trisna Pratama
NRP 07111540000108

Advisor
Astria Nur Irfansyah, ST., M.Eng., Ph.D
Fajar Budiman, S.T., M.Eng.

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2019

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Lokalisasi Suara Menggunakan FPGA Berbasis Komputasi Stokastik**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, 10 Juli 2019

Rifqi Yunus Trisna P.
NRP. 0711 15 40000 108

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

LOKALISASI SUARA MENGGUNAKAN FPGA BERBASIS KOMPUTASI STOKASTIK

TUGAS AKHIR

Diajukan untuk Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui,

Dosen Pembimbing 1	Dosen Pembimbing 2
Astria Nur Irfansyah, ST., M.Eng., Ph.D.	Fajar Budiman, S.T., M.Eng.
NIP : 198103252010121002	NIP : 198607072014041001

KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI ELEKTRO
DEPARTEMEN TEKNIK ELEKTRO

SURABAYA
JULI, 2019

Halaman ini sengaja dikosongkan

LOKALISASI SUARA MENGGUNAKAN FPGA BERBASIS KOMPUTASI STOKASTIK

Nama : Rifqi Yunus Trisna Pratama
Pembimbing : Astria Nur Irfansyah, ST., M.Eng., Ph.D
Fajar Budiman, S.T., M.Eng.

ABSTRAK

Komputasi stokastik adalah teknik komputasi digital alternatif yang tidak menggunakan representasi bilangan digital dalam bentuk biner, melainkan memanfaatkan representasi bilangan stokastik. Tujuannya adalah penyederhanaan rangkaian komputasi, di mana komputasi bisa dilakukan menggunakan gerbang logika dasar.. Pada aplikasi-aplikasi tertentu, komputasi stokastik menunjukkan tingkat toleransi error yang tinggi namun tetap memberikan hasil yang lebih memuaskan dibandingkan metode komputasi biner konvensional yang mengharuskan tingkat error pada data yang sangat rendah untuk memberikan pembacaan yang akurat.

Dalam tugas akhir ini, teknik berbasis komputasi stokastik dieksplorasi dan diimplementasikan untuk aplikasi sistem detektor arah suara. *Output* dari sistem yang dibuat adalah lokalisasi suara dengan penggunaan rangkaian komputasi stokastik yang lebih sederhana dari sistem komputasi biner. Dengan begitu, dapat dimungkinkan untuk menempatkan rangkaian ini tanpa memakan terlalu banyak tempat, mengingat kompleksitas rangkaian yang jauh lebih rendah dari sistem lokalisasi suara yang sudah ada. Salah satu fitur kunci yang akan dikonversi pada tugas akhir ini adalah konsep pengukuran selisih waktu kedatangan sinyal audio (interaural time difference, ITD). Sistem komputasi untuk ITD dapat memanfaatkan *cross-correlation*. Hasil dari *cross-correlation* akan dimasukkan pada proses *time-delay analysis* agar dapat ditemukan *delay* antara kedua sinyal *input*. Hasil yang didapatkan untuk *cross-correlation* memiliki hasil yang cukup konsisten. Dengan hasil *cross-correlation* tersebut, didapatkan pergeseran dan *delay* sinyal yang masuk pada kedua *input*.

Kata Kunci: Lokalisasi suara, FPGA, Komputasi Stokastik, Cross-Correlation

Halaman ini sengaja dikosongkan

SOUND LOCALIZATION USING FPGA BASED ON STOCHASTIC COMPUTATION

Name : Rifqi Yunus Trisna Pratama
Supervisor : Astria Nur Irfansyah, ST., M.Eng., Ph.D
Fajar Budiman, S.T., M.Eng.

ABSTRACT

Stochastic computation is an alternative digital computation technique that does not use representation of digital numbers in binary form, but utilizes stochastic number representations. The goal is to simplify the computational circuit, where computation can be done using basic logic gates. In certain applications, stochastic computation shows a high level of error tolerance but still gives more satisfying results than conventional binary computation methods that require very low error rates in order to provide accurate readings

In this final project, stochastic computation based techniques are explored and implemented for sound localization applications. The output of the system created is a sound direction localization with the use of a simpler stochastic computation circuit than a binary computing system. That way, it can be possible to place this circuit without consuming too much space, given the complexity of the circuit is much lower than the existing sound localization system. One of the key features that will be converted in this final project is the concept of measuring the difference in arrival time of audio signals (interaural time difference, ITD). The computing system for ITD can utilize the cross-correlation. The results of the cross-correlation will be processed in time-delay analysis so that delay can be found between the two input signals. The results obtained for the cross-correlation have fairly consistent results. With the results of the cross-correlation, a shift and delay signal can be computed from both inputs.

Keywords: Sound localization, FPGA, Stochastic Computation, Cross-Correlation

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji dan syukur kepada Allah SWT atas berkat dan hikmat yang diberikan, penulis dapat menyelesaikan laporan tugas akhir dengan judul "**Lokalisasi Suara Menggunakan FPGA Berbasis Komputasi Stokastik**", sebagai salah satu persyaratan dalam menyelesaikan pendidikan program Strata-Satu di Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember.

Dalam penyusunan dan penyelesaian laporan Tugas Akhir ini penulis mendapatkan banyak sekali doa, bantuan dan dukungan dari berbagai pihak. Untuk itu penulis mengucapkan terima kasih yang sebesar – besarnya kepada:

1. Orang tua beserta keluarga penulis, yang tak henti-hentinya memberikan semangat, dukungan dan kasih sayang yang luar biasa kepada penulis.
2. Bapak Astria Nur Irfansyah, ST., M.Eng., Ph.D. dan Bapak Fajar Budiman, S.T., M.Eng selaku dosen pembimbing atas gagasan topik tugas akhir serta bimbingan dan arahan untuk penulis selama mengerjakan tugas akhir.
3. Seluruh dosen bidang studi elektronika Departemen Teknik Elektro ITS.
4. Seduluran Kopi Lokal Gresik yang sudah membantu dalam dukungan moral dan materiil selama masa kuliah.
5. Linggar Ajeng Y.M. yang telah turut membantu dalam pengerjaan dan memberikan semangat.
6. Rekan-rekan bidang studi elektronika yang tidak dapat disebutkan satu-persatu yang telah membantu dan memberikan semangat kepada penulis selama menjalani perkuliahan di Departemen Teknik Elektro ITS.

Penulis menyadari bahwa masih banyak yang dapat dikembangkan pada tugas akhir ini. Oleh karena itu penulis menerima setiap masukan dan kritik yang diberikan. Semoga tugas akhir ini dapat memberikan manfaat bagi banyak pihak.

Surabaya, 10 Juli 2019

Rifqi Yunus Trisna Pratama

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

1.1 Latar Belakang

Cukup banyak penelitian mengenai lokalisasi suara dengan menggunakan beberapa array mikrofon yang telah dipublikasikan. Berbagai macam metode telah diaplikasikan untuk aplikasi ini, dan seluruh metode tersebut menggunakan sistem komputasi konvensional biner. Pada penelitian ini, akan digunakan sistem komputasi alternatif yang dirasa akan cocok dengan kondisi sinyal dan data yang akan diakuisisi oleh sensor. Dengan sistem alternatif tersebut, sinyal yang berisi data dengan *noise-noisenya*, diharapkan dapat diproses dengan memanfaatkan karakteristik dari sistem komputasi alternatif tersebut yang dinilai lebih tahan terhadap error-error yang bermunculan pada proses akuisisi data maupun pre-processing yang dilakukan pada data terkait.

Pada penelitian ini, pemrosesan audio memanfaatkan metode komputasi stokastik. Suatu metode yang sempat ditinggalkan selama lebih dari 50 tahun, namun kembali diperhitungkan karena kesederhanaan rangkaiannya serta kemampuannya untuk bertahan dalam kondisi yang memiliki cukup banyak *noise*[1]. Selain itu, kesederhanaan rangkaiannya yang menggunakan gerbang logika sederhana untuk melakukan komputasi perkalian dan penjumlahan memungkinkan proses operasi perhitungan diproses lebih cepat. Berbeda dengan sistem komputasi biner yang memerlukan proses yang dapat dibilang jauh lebih rumit dibandingkan sistem komputasi stokastik. Sistem yang diaplikasikan diharapkan akan berdampak positif pada kompleksitas rangkaian. Salah satu penelitian yang menjadi acuan adalah sistem yang menggunakan mikrokontroller berkecepatan tinggi untuk menghitung sistem *cross-correlation*[2].

Sistem yang akan dibuat diharapkan dapat memperbaiki efisiensi baik dari segi besar rangkaian dan *resource* yang digunakan serta dari konsumsi daya maupun kecepatan. Dengan memanfaatkan kecilnya *area* rangkaian yang dibuat, sistem *prototype* yang dibuat dapat diproduksi masal dengan menggunakan biaya yang lebih rendah mengingat sistem komputasi stokastik memiliki kesederhanaan rangkaian yang cukup baik.

1.2 Perumusan Masalah

Perumusan masalah pada Tugas Akhir ini adalah:

1. Apakah metode audio processing menggunakan komputasi stokastik dapat digunakan untuk mengurangi kompleksitas rangkaian sistem?
2. Bagaimana cara implementasi metode tersebut pada sistem?

1.3 Batasan Masalah

Batasan masalah dalam tugas akhir ini adalah:

1. Objek pengujian berupa suara kontinyu yang berulang
2. Perhitungan stokastik akan dilakukan pada sebagian perhitungan.

1.4 Tujuan

Diharapkan dengan adanya pengembangan tugas akhir ini, bisa memberikan :

1. Sistem dapat menggunakan metode komputasi stokastik sebagai alternatif dari metode konvensional.
2. Metode dapat diimplementasikan menggunakan Field Programmable Gate Array (FPGA) untuk tujuan prototyping, dan dapat dijadikan sebagai IC tersendiri apabila desain sudah final dan teruji.

1.5 Metodologi

Dalam pengerjaan tugas akhir ini digunakan metodologi sebagai berikut:

1. Studi Literatur

   Studi literatur akan berisi pengumpulan serta pengkajian teori, data dan penelitian yang dianggap relevan dan terpercaya untuk mendukung keabsahan tugas akhir ini. Literatur yang digunakan akan memiliki batasan-batasan tertentu. Yaitu, literatur yang digunakan harus bersumber dari *paper,* jurnal, buku, maupun artikel yang berasal dari badan pemerintahan atau institusi akademik terpercaya.
2. Perancangan dan Pembuatan Rangkaian Eksternal

   Perancangan rangkaian sensor dibutuhkan untuk menentukan agar bisa didapatkan data yang optimal untuk lokalisasi suara. Akan diperlukan pengambilan data-data, agar didapatkan karakteristik sensor mikrofon dan didapatkan jumlah

mikrofon serta posisi yang tepat.

Rangkaian pre-amp dibutuhkan untuk memperkuat sinyal audio yang didapatkan dari mikrofon. Hal ini sangat dibutuhkan dikarenakan sinyal audio mentah yang dikeluarkan oleh microphone sangat lemah, sehingga diperlukan rangkaian external untuk memperkuat sinyal *input* yang nantinya akan diproses oleh FPGA.

Setelah dilakukan perancangan dan simulasi, Dilakukan tahap pembuatan rangkaian secara keseluruhan. Akan dilakukan proses desain pcb, penmbelian komponen serta proses *assembly.*

3. Sintesis Rangkaian Internal FPGA

   Setelah rangkaian eksternal dapat dipastikan dapat bekerja dengan baik. Selanjutnya, akan dilakukan sintesis rangkaian sistem berbasis komputasi stokastik. Salah satu rangkaian penting yang akan menjadi salah satu inti utama adalah *Linear Feedback Shift Register*, selaku *pseudo random generator*. Serta rangkaian *audio processing* yang akan menjadi rangkaian *processor* dan penentu arah suara. Yang akan didesain dan dipersiapkan lebih detail saat pengerjaan.

4. Pengujian Sistem

   Merupakan tahap pengujian keseluruhan sistem, apabila masih terdapat kesalahan pada sistem sehingga sistem tidak dapat berjalan dengan baik dan perlu revisi pada desain rangkaian atau sintesis pada FPGA. Beberapa tahap pengujian berupa pengujian rangkaian sensor sebelum dilakukan penguatan. Lalu pengujian rangkaian eksternal secara keseluruhan, berupa sensor dan rangkaian pre-amp. Dilanjutkan dengan pengujian sistem secara keseluruhan untuk menghasilkan data final, yang merupakan tujuan dari sistem ini.

5. Analisa Data dan Evaluasi

   Pada tahap ini, akan dilakukan Analisa terhadap data yang didapatkan. Sehingga didapatkan karakteristik dari *hardware* yang telah dirancang dan dibuat. Analisa dilakukan khususnya pada rangkaian eksternal, apakah sudah memenuhi kriteria untuk dilakukan pemrosesan pada FPGA. Lalu dilakukan Analisa terhadap data yang dihasilkan oleh keseluruhan sistem, apakah sudah dianggap memuaskan atau tidak. Lalu dilakukan evaluasi serta revisi desain apabila diperlukan

6. Penyusunan Laporan

   Penyusunan laporan akan dilakukan seiring dengan tahap-tahap lainnya. Isinya berkaitan dengan tugas akhir yang dikerjakan, meliputi pendahuluan, studi literatur, perancangan dan pembuatan sistem, pengujian dan Analisa serta penutup.

1.6 Sistematika Penulisan

Dalam buku tugas akhir ini, pembahasan mengenai sistem yang dibuat terbagi menjadi lima bab dengan sistematika penulisan sebagai berikut:

- BAB I: PENDAHULUAN

  Bab ini meliputi penjelasan latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, sistematika penulisan, dan relevansi.
- BAB II: TINJAUAN PUSTAKA DAN TEORI PENUNJANG

  Bab ini menjelaskan tentang teori penunjang dan literatur yang dibutuhkan dalam pengerjakan tugas akhir ini. Dasar teori yang menunjang meliputi metode lokalisasi suara, konsep dasar komputasi stokastik dan diagram cara kerja sistem. Bagian ini memaparkan mengenai beberapa teori penunjang dan beberapa literatur yang berguna bagi pembuatan Tugas Akhir ini.
- BAB III: PERANCANGAN SISTEM

  Bab ini menjelasakan secara rinci tentang perencanaan sistem baik *hardware* (*hardware*) maupun *software* (*software*) sistem lokalisasi suara yang dikerjakan.
- BAB IV: PENGUJIAN

  Pada bab ini akan menjelaskan hasil uji coba sistem beserta analisanya.
- BAB V: PENUTUP

  Bagian ini merupakan bagian akhir yang berisikan kesimpulan yang diperoleh dari pembuatan Tugas Akhir ini, serta saran-saran untuk pengembangan lebih lanjut.

1.7 Relevansi

Lokalisasi Suara Menggunakan FPGA Berbasis Komputasi Stokastik merupakan suatu metode baru, memanfaatkan metode komputasi alternatif yang sempat ditinggalkan namun kembali populer dalam beberapa tahun terakhir. Hal ini disebabkan oleh tingginya tingkat toleransi error yang mana akan sangat berguna ketika digunakan untuk memproses data yang memiliki banyak *noise*, seperti pada sinyal suara.

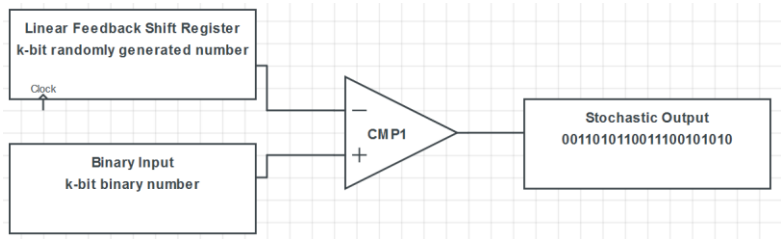
Halaman ini sengaja dikosongkan

BAB II
DASAR TEORI

Pada Bab II, dasar teori penunjang akan menjelaskan mengenai bagian-bagian yang berhubungan dengan sistem yang akan dibuat pada tugas akhir ini beserta teori-teori yang menunjang penggunaannya. Sedangkan tinjauan pustaka dalam bab ini menjelaskan mengenai penelitian, sistem dan metode yang berhubungan dengan tugas akhir ini yang telah diimplementasikan dan dipublikasikan oleh penulis-penulis sebelumnya.

2.1 Angka Stokastik

Angka stokastik merupakan format angka digital alternatif dari format angka biner konvensional. Umumnya, setiap bit dari N-bit angka stokastik, X dipilih sebagai 1 yang dipilih secara acak dengan probabilitas px, dan X merupakan angka yang dibuat dari rangkaian logika konvensional. Untuk mengkonversi angka biner menjadi angka stokastik, akan dibutuhkan *random number source* (RNS) dan komparator sebagai alat pembanding nilai keduanya seperti pada gambar 2.1.

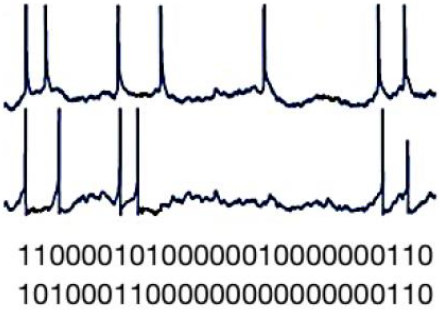


Gambar 2.1 Generator Angka Stokastik [1]

Sistem bilangan stokastik merupakan salah satu alternatif yang sempat digunakan pada tahun 1960-an. Sistem bilangan ini kembali dipertimbangkan dikarenakan potensinya menyusul berkembangngya teknologi nano beserta aplikasinya seperti ECC *decoding* dan pemrosesan citra biomedik. Namun beserta dengan potensi tersebut, terdapat banyak tantangan dan kendala yang harus diatasi sebelum sistem bilangan ini dapat dimanfaatkan dengan maksimal.

Nilai dari sebuah angka stokastik ditentukan oleh seberapa banyak

digit 1 muncul pada satu seri N-bit angka stokastik. Pengkodean semacam ini juga sering ditemukan pada sistem syaraf biologis yang biasanya ada pada hewan dan manusia.[3]. Perbandingan keduanya digambarkan pada gambar 2.2.



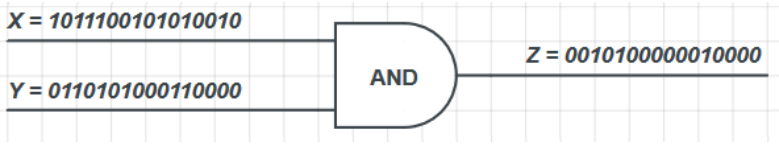
Gambar 2.2 Rentetan Spike pada Jaringan Syaraf Biologis serta Representasinya pada Format Angka Stokastik [1]

2.2 Komputasi Stokastik

Komputasi stokastik merupakan metode non-konvensional yang memperlakukan data sebagai probabilitas. Format angka yang digunakan memiliki format khusus, yaitu angka stokastik. Angka ini dibuat dan diproses oleh rangkaian logika konvensional. Beberapa contoh aplikasi rangkaian logika sebagai operator bilangan stokastik adalah:

2.2.1 AND Gate

AND gate dapat digunakan sebagai proses perkalian pada sistem komputasi stokastik. AND gate dianalogikan sebagai pengali probabilitas, sistem yang mendasari sistem komputasi stokastik.



Gambar 2.3 Operasi Perkalian Stokastik Menggunakan Gerbang AND [1]

Seperti yang dilihat pada gambar 2.3, AND gate mengaplikasikan persamaan probabilitas berupa

$$p(Z) = p(X) \times p(Y) \quad (2.1)$$

Mengacu pada persamaan 2.1 operasi pada gambar 2.3 dapat dibuat perhitungan berupa

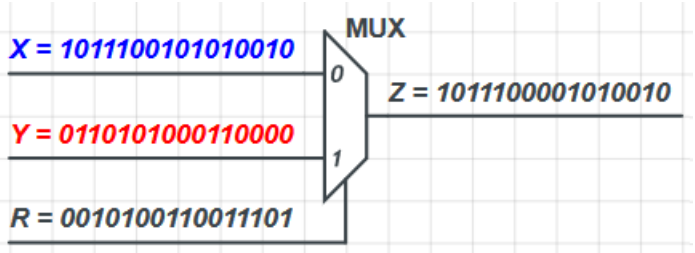
$p(X) = {}^{8}/_{16}$
$p(Y) = {}^{6}/_{16}$

$p(Z) = p(X) \times p(Y) = {}^{8}/_{16} \times {}^{6}/_{16} = {}^{3}/_{16}$

Mengacu pada perhitungan yang telah dibuat, secara teknis, perkalian dapat dilakukan hanya dengan menggunakan satu gerbang logika AND yang jauh lebih sederhana dibandingkan dengan sistem komputasi konvensional.

AND gate digunakan sebagai pengali probabilitas. Pola output dari operator ini sangat bergantung pada pola bit *input* X dan Y. Kedua *input* diharuskan independen satu sama lain atau tidak memiliki korelasi agar px dan py dapat dianggap sebagai probabilitas yang independen satu sama lain, yang menghasilkan output Z yang sesuai dan menghindari error yang diakibatkan *input* yang memiliki korelasi tinggi antara satu sama lain.

2.2.2 *Multiplexer Gate*



Gambar 2.4 Operasi Penjumlahan Stokastik Menggunakan Multiplexer [1]

Pada operasi penjumlahan stokastik pada gambar 2.4, nilai *input* penjumlahan kedua *input* dipilih secara acak berdasarkan selektor yang memiliki probabilitas 0,5. Multiplexer menngaplikasikan fungsi Boolean seperti persamaan 2.2.

$$z = (X\ AND\ R)\ OR\ (Y\ AND\ R) \quad (2.2)$$

dimana X dan Y merupakan data *input* utama dan R merupakan selector berupa deret acak dengan probabilitas 0,5. Setengah output Z pada gambar 2.4 berasal dari *input* X (biru) dan setengah lainnya berasal dari *input* Y (merah) sesuai dengan *input* selector R dengan probabilitas 0,5. Maka, dapat dibuat persamaan 2.3,

$$p(Z) = 0.5\big(p(X)\big) + 0.5(p(Y)) \quad (2.3)$$

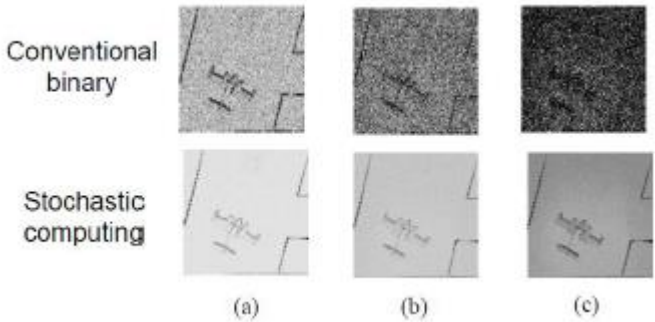
Serta mengacu pada persamaan 2.1 dan gambar 2.4, dapat dibuat perhitungan berupa

$$p(X) = {}^{8}/_{16}$$
$$p(Y) = {}^{6}/_{16}$$
$$p(R) = {}^{8}/_{16}$$

$$p(Z) = {}^{1}/_{2}\,(p(X) + p(Y)) = {}^{1}/_{2}\left({}^{8}/_{16} + {}^{6}/_{16}\right) = {}^{7}/_{16}$$

Kesederhanaan rangkaian-rangkaian operator tersebut memungkinkan dibentuknya sistem parallel untuk mempercepat proses perhitungan, dengan konsekuensi sedikit meningkatnya konsumsi daya pada sistem.

Panjang N-bit angka stokastik, mempengaruhi langsung tingkat error dari hasil operasi angka stokastik, namun dengan bertambahnya jumlah bit akan bertambah pula waktu proses yang dibutuhkan. Seiring dengan waktu maka akan berbanding lurus dengan konsumsi daya dari sistem. Tingkat presisi yang rendah juga merupakan salah satu faktor yang menyebabkan komputasi stokastik sempat ditinggalkan. Pada tingkat presisi 8 bit pada sistem kompitasi biner, diperlukan 256 bit bilangan stokastik dan jumlahnya harus digandakan seiring bertambahnya bit pada bilangan biner. Namun pada beberapa aplikasi tertentu, komputasi stokastik terbukti dapat mengungguli komputasi biner konvensional pada data yang memiliki tingkat *soft error* yang cukup tinggi, salah satu contoh yang paling terlihat ada pada image processing[1], digambarkan pada gambar 2.5.



Gambar 2.5 Hasil Edge Detection untuk Kedua Metode dengan Level *Noise* (a) 5%, (b) 10% dan (c) 20%. [1]

Sistem komputasi stokastik yang mengandalkan tingkat keacakan data sangat tahan kepada *soft error*, namun hal ini juga berarti bahwa komputasi ini tidak memungkinkan penggunaannya pada aplikasi yang membutuhkan tingkat akurasi yang tinggi dan mengharuskan tingkat error yang sangat rendah pada proses komputasinya. Dikarenakan ketika komputasi stokastik dipaksakan untuk memiliki tingkat error yang rendah, akan berdampak cukup buruk pada waktu pemrosesan data[1].

2.3 *Field Programmable Gate Array*

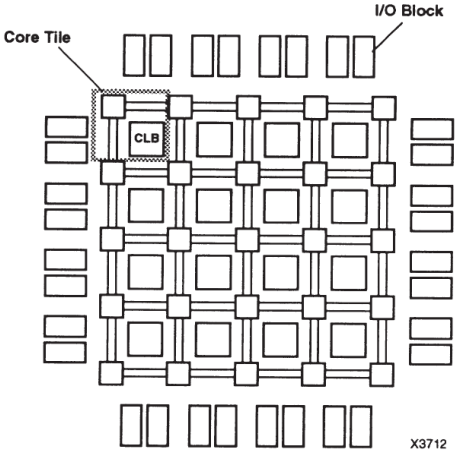
Field Programmable Gate Array digunakan sebagai pemroses utama dari sistem yang dibuat. FPGA merupakan salah satu bentuk rangkaian terintegrasi (*Integrated Circuit*) yang didesain agar dapat dikonfigurasi setelah keluar dari proses manufaktur. FPGA pada umumnya dikonfigurasi menggunakan bahasa *Hardware Description Language* (HDL), serupa dengan Bahasa yang digunakan untuk penggunaan *application-specific integrated circuit* (ASIC).

FPGA pertama kali dikenalkan pada tahun 1985 oleh Xilinx. Seiring waktu, FPGA semakin dikembangkan oleh beberapa perusahaan lainnya seperti Altera, QuickLogic dan beberapa perusahaan lainnya.

FPGA memiliki susunan programmable *logic blocks* dan hierarki interkoneksi diantara blok-blok tersebut, sehingga dapat dihubungkan antara satu dan lainnya untuk membentuk sistem yang diinginkan (Gambar 2.6). *Logic block* dapat dikonfiguraasi sehingga membentuk fungsi kombinansional atau sekuensial kompleks, atau sekedar menghasilkan gerbang logika sederhana seperti OR atau AND.

Interkoneksi dapat deprogram sehingga satu logic block dapat dikoneksikan dengan *logic block*.[4]

FPGA pada umumnya digunakan sebagai alat untuk memfasilitasi *prototyping* sebelum rangkaian diproduksi masal. Beberapa aplikasi lain dapat berupa sistem yang mungkin perlu direvisi setelah dijual ke pihak konsumen untuk perbaikan maupun pembaharuan. Sehingga sangat memungkinkan dalam implementasi sistem yang memanfaatkan gerbang logika sederhana sebagai media komputasi. FPGA juga merupakan dapat memperlihatkan komponen-komponen yang digunakan pada suatu sistem, sehingga dapat dibandingkan antara penggunaan rangkaian logika pada sistem komputasi biner dan sistem komputasi stokastik.



Gambar 2.6 Konsep Susunan *Hardware* FPGA[4]

2.4 *DE10-Nano Development Board*

DE10-Nano *Development Board* (Gambar 2.7) merupakan *platform hardware design* yang dibuat berdasarkan *System-on-Chip* (SoC) FPGA Intel. Memiliki total 80 pin GPIO berbasis tegangan 2.5V hingga 3.3V bergantung pada konfigurasi dari FPGA. Menggunakan *power supply* terpisah dari USB untuk memfasilitasi arus yang mungkin dibutuhkan oleh sistem.

Dengan memanfaatkan USB-Blaster II untuk *onboard programming*, akan sangat memudahkan untuk melakukan pemrograman tanpa menggunakan perantara alat lain. Serta dengan tambahan *input slide*

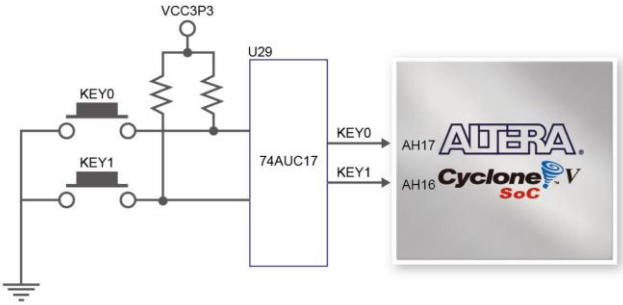
switch dan *push button*, serta dengan *output* LED, akan sangat memudahkan untuk mengetahui *output* serta memberi *input* tanpa perlu rangkaian tambahan.

Gambar 2.7 DE10-Nano *Development Board*

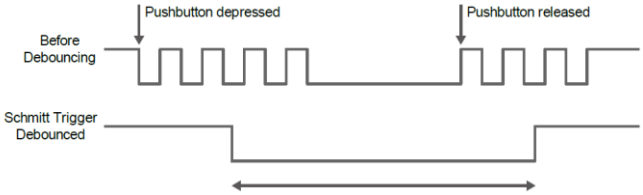
DE10-Nano memiliki berbagai fitur yang membantu agar *prototyping* pada FPGA menjadi lebih mudah, beberapa diantaranya yang akan digunakan pada sistem adalah:

2.4.1 Push-button, Switch *dan* LED

Board DE10-Nano memiliki dua *push-button* yang tersambung pada FPGA seperti yang terlihat pada gambar 2.8. Rangkaian *Schmitt trigger* digunakan untuk menanggulangi kemungkinan *switch debounce* (Gambar 2.9) pada kedua *push-button* yang digunakan. Dua *push-button* dengan nama standar KEY0 dan KEY1 yang telah dilewatkan rangkaian *Schmitt Trigger* tersambung langsung dengan Cyclone V SoC FPGA. *Push-button* dirangkai dengan resistor *pull-up*. Setelah melalui proses *debouncing*, *push-button* dapat digunakan sebagai *clock* maupun *reset* dalam rangkaian suatu sistem.



Gambar 2.8 Koneksi antara *Push-button* dan Cyclone V SoC FPGA



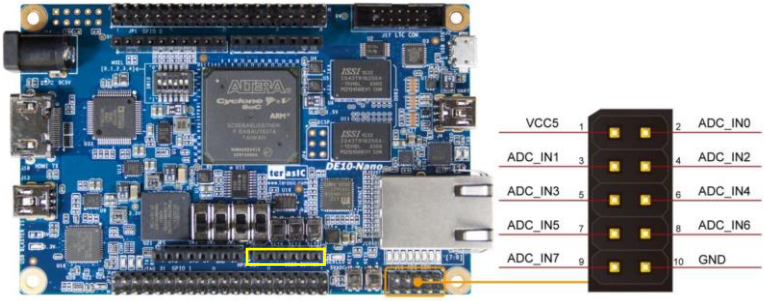
Gambar 2.9 Skema *Switch Debouncing*

DE10-Nano memiliki 4 buah *slide switch* yang umumnya digunakan sebagai *input data* pada rangkaian sistem (Gambar 2.10). Setiap *switch* tersambung langsung pada FPGA.

Gambar 2.10 Koneksi antara *Slide Switch* dan Cyclone V SoC FPGA

2.4.2 *Analog-to-Digital Converter*

ADC digunakan sebagai *converter* sinyal analog menjadi sinyal digital sehingga dapat diproses oleh sistem FPGA yang merupakan *processor* digital. DE10-Nano menggunakan IC LTC2308 yang merupakan IC ADC dengan 8 channel yang dapat diakses melalui sistem komunikasi SPI. ADC DE10-Nano memiliki *header* khusus berisi *input* 8 channel beserta supply 5v dan terminal ground.



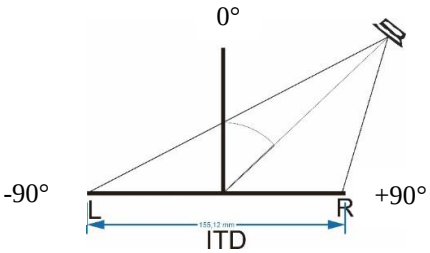
Gambar 2.11 *Header* ADC pada *Board* DE10-Nano

Analog *input* ini juga tersambung pada analog *input* pada header Arduino yang tersedia pada DE10-Nano yang ditandai oleh kotak kuning pada gambar 2.11

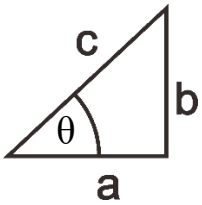
2.5 Lokalisasi Suara

Lokalisasi suara adalah kemampuan dari pendengar untuk mendeteksi asal suara berdasarkan arah dan/atau jarak [5]. Prinsip lokalisasi suara paling mendasar adalah dengan mengukur selisih waktu kedatangan sinyal *audio* (*interaural time difference*, ITD) atau dengan mengukur level dari sinyal audio tersebut (*interaural intensity difference*, IID). Untuk sinyal sederhana seperti sebuah sinus tunggal, selisih waktu ini bisa didapatkan dari beda fase kedua gelombang, ini juga bisa disebut interaural phase difference. Untuk gelombang akustik secara umum, ITD bisa dikomputasi dengan memanfaatkan *cross-correlation*. Untuk tugas deteksi arah suara yang lebih sulit seperti dalam ruangan yang ada pantulan suara, teknik lain yang lebih kompleks atau memanfaatkan bentuk yang ditemukan pada sistem biologis (seperti *cochlea*, atau *neural network*) yang telah dilaporkan di literatur.

Interaural Time Difference dalam konteks pendengaran manusia maupun hewan merupakan selisih waktu datang suara diantara kedua telinga. ITD terjadi karena kedua "telinga" diposisikan dengan jarak d antara satu sama lain. ITD merupakan salah satu metode sensor *binaural* yang memanfaatkan selisih *timing* suara dari tiap sensor yang tersedia. Ketika sinyal dihasilkan pada bidang horizontal, sudut relatif terhadap sensor disebut sebagai azimuth, di mana 0 derajat (0°) azimuth ada di tepat di depan sensor, seperti yang digambarkan pada gambar 2.12.



Gambar 2.12 Perhitungan Sudut *Azimuth*.



Gambar 2.13 Perhitungan Sudut Menggunakan Prinsip Trigonometri

ITD digunakan dengan pertimbangan bahwa pergeseran waktu dan fasa dapat dideteksi menggunakan metode *cross-correlation*. Pada frekuensi rendah *timing* ITD didapatkan dengan mencari delay fasa, namun pada frekuensi tinggi *timing* didapatkan dengan mencari *group delay*. Perhitungan azimuth pada ITD memanfaatkan prinsip trrigonometri sederhana seperti pada gambar 2.13 dengan memanfaatkan nilai c sebagai jarak kedua mikrofon dan nilai a sebagai jarak yang dicari dari kecepatan suara yang dikalikan dengan *delay* antara kedua mikrofon yang didapatkan dari *cross-correlation* seperti pada persamaan 2.3.

$$length = t \times V_{sound} = (\Delta \times \sigma) \times V_{sound} \qquad (2.3)$$

Nilai t pada persamaan 2.3 adalah waktu jeda yang digunakan untuk mencari nilai sudut dimana Δ merupakan frekuensi sampling per-indeks dan σ merupakan *delay* yang didapatkan dari indeks maksimum perhitungan *cross-correlation*, nilai V_{sound} merupakan konstanta sebesar 384 m/s. Setelah itu, didapat nilai a dan c yang hasilnya dapat digunakan untuk menghitung sudut arah sumber suara memanfaatkan rumus trigonometri sesuai persamaan 2.4 dan 2.5.

$$\sin\theta = \frac{a}{c} \therefore \theta = \sin^{-1}\frac{a}{c} \tag{2.4}$$

$$\theta = \sin^{-1}\frac{(\Delta\times\sigma)\times V_{sound}}{c} \tag{2.5}$$

2.6 *Hardware Description Language*

Pada kurun waktu yang cukup lama, bahasa pemrograman seperti FORTRAN, Pascal dan C digunakan untuk mendeskripsikan program komputer yang bersifat sekuensial. Sama halnya pada bidang desain digital, desainer merasa sistem yang mirip perlu dikembangkan sehingga dapat digunakan bahasa yang standar untuk mendeskripsikan rangkaian digital. Oleh karena itu, dibuatlah standar berupa *Hardware Description Language* (HDL)[6].

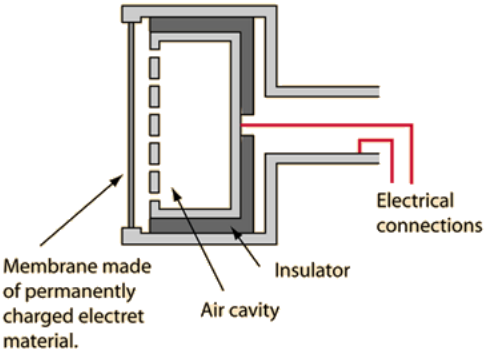
Verilog *Hardware Description Language* merupakan salah satu dari dua jenis HDL yang digunakan di dunia. Verilog HDL dipilih sebagai bahasa yang digunakan untuk mendesain sistem komputasi utama pada FPGA. Verilog-HDL merupakan salah satu dari dua bahasa deskripsi *hardware* (*Hardware Description Language*) yang digunakan pada sistem FPGA. Verilog memiliki kemiripan *syntax* dengan bahasa pemrograman C yang sudah banyak digunakan sebagai bahasa pemrograman dalam bidang pengembangan *software*. Seperti bahasa C, Verilog merupakan bahasa pemrograman yang *case sensitive*. Alur kontrol yang disediakan (*if/else, for, while, case,* dll) memiliki kemiripan dengan C. Perbedaan dari kedua bahasa tersebut salah satunya ada pada deklarasi variabel pada Verilog yang membutuhkan lebar bit pada deklarasi variabel, walaupun hal ini dapat dilewati dan Verilog akan mengasumsikan ukuran terbesar yaitu sebesar 32 bit, selain dari itu Verilog tidak menggunakan kurung kurawal sebagai penanda demarkasi blok prosedural namun menggunakan pasangan "*begin/end*" sebagai pengganti fungsi kurung kurawal ({}). Kemiripannya dengan bahasa C yang merupakan bahasa

pemrograman yang banyak digunakan hingga saat ini sangat memudahkan bagi pemula untuk mempelajari bahasa ini.

HDL cukup popular dalam penggunaan awal dalam proses *logic verification*, namun desainer masih diharuskan mengartikan bahasa tersebut dan merancang skematik rangkaian logika secara manual. Perkembangan *logic synthesis* pada tahun 1980-an mengubah metodologi desain rangkaian logika secara radikal. Rangkaian digital dapat dideskripsikan pada *register transfer level* (RTL) dengan menggunakan HDL. Sehingga, desainer hanya perlu mendeskripsikan bagaimana aliran data antar register dan bagaimana sistem memproses data tersebut. Interkoneksi dan detail mengenai gerbang-gerbang yang digunakan untuk menyusun rangkaian akan dibuat berdasarkan deskripsi RTL oleh *logic synthesis tools*.

2.7 Mikrofon

Mikrofon merupakan salah satu transduser yang merubah besaran fisik menjadi besaran listrik, dalam kasus ini adalah sinyal suara menjadi sinyal listrik berupa tegangan. Mikrofon memiliki berbagai macam jenis, beberapa contohnya adalah mikrofon kristal, mikrofon elektrodinamik dan mikrofon kondensor. Mikrofon yang digunakan dalam sistem ini adalah mikrofon kondensor elektret.



Gambar 2.14 Mikrofon Kondensor Elektret

Mikrofon kondensor biasa juga disebut sebagai mikrofon kapasitor atau mikrofon elektrostatis. Diafragma berfungsi sebagai salah

satu plat dalam kapasitor keping sejajar. Diafragma pada mikrofon kondensor terbuat dari bahan yang ringan sehingga dapat mendeteksi dan beresonansi dengan getaran di udara. Getaran yang terjadi pada diafragma akan mengubah jarak antara kedua keping sehingga tegangan yang disimpan oleh mikrofon berubah-ubah mengikuti perubahan kapasitansi.

Mikrofon electret seperti pada gambar 2.14 merupakan mikrofon condenser yang telah terpolarisasi secara permanen. Hal ini dapat terjadi karena adanya substansi elektret yang diaplikasikan pada mikrofon kondensor elektret. Substansi elektret menggantikan fungsi muatan eksternal yang dibutuhkan oleh mikrofon kondensor konvensional yang membutuhkan *input* tegangan cukup tinggi. Kemudahan manufaktur dan kinerja yang bagus membuat mikrofon kondensor elektret banyak digunakan untuk berbagai macam aplikasi.

2.8 Pre-amplifier

Preamplifier merupakan perangkat elektronik yang memperkuat sinyal *input* yang lemah sehingga cukup kuat dan toleran terhadap *noise* untuk selanjutnya dapat diproses lanjut, atau dikirimkan langsung ke *power amplifier* dan selanjutnya ke *loudspeaker*. Preamplifier pada umumnya digunakan untuk memperkuat sinyal dari sensor analog seperti mikrofon. Oleh karena itu, untuk mengurangi efek *noise* dan interferensi, rangkaian preamplifier ditempatkan dekat dengan sensor yang outputnya akan dikuatkan.

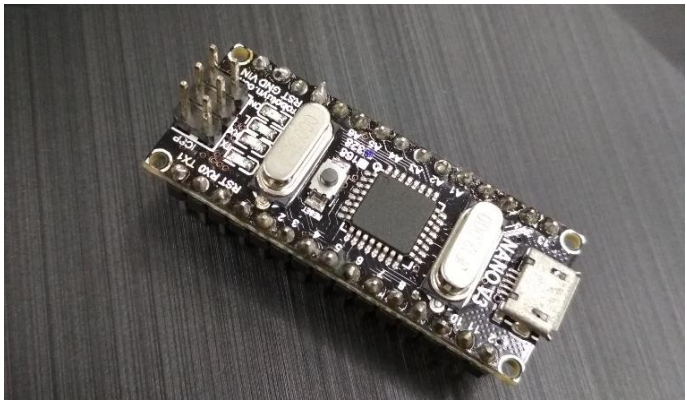
Preamplifier merupakan penguat tegangan yang akan menguatkan pembacaan sensor sehingga dapat diproses oleh mikrokontroller maupun prosesor lainnya. Preamplifier digunakan untuk mengeliminasi kemungkinan interferensi dan *noise* yang timbul pada transmisi antara sensor dan prosesor. Pada aplikasi di lapangan, pre-amplifier memiliki begitu banyak variasi, mulai dari berbagai macam model skematik rangkaian dan berbagai macam komponen yang digunakan. Namun berdasarkan komponen utama, preamplifier pada umumnya menggunakan satu dari dua komponen yaitu menggunakan transistor atau menggunakan IC Op-amp.

2.9 Arduino

Arduino adalah *platform open-source* yang digunakan untuk membangun proyek elektronik. Arduino terdiri dari papan sirkuit fisik yang dapat diprogram (sering disebut sebagai mikrokontroler) dan *software*, atau IDE (*Integrated Developed Environment*) yang berjalan di

komputer, yang fungsinya untuk menulis dan mengunggah kode komputer ke papan fisik. *Platform* Arduino telah menjadi sangat populer di kalangan orang-orang yang baru memulai dengan elektronik, dan dengan kelebihan-kelebihan serta alasan yang baik. Tidak seperti kebanyakan papan sirkuit yang dapat diprogram sebelumnya, Arduino tidak perlu *hardware* terpisah (disebut programmer) untuk memuat kode baru ke papan, cukup menggunakan kabel USB. Selain itu, Arduino IDE menggunakan versi sederhana C ++, membuatnya lebih mudah untuk belajar memprogram. Selain itu, komunitas Arduino yang sudah sangant berkembang menyediakan begitu banyak dokumentasi maupun *library* yang sangat mempermudah dan mempersingkat proses-proses pengaksesan modul dan perangkat tambahan.

Arduino Nano adalah salah satu *development board* Arduino berbasis ATmega 328. Memiliki 14 pin *input* dan *output* digital, di mana 2 pin digunakan sebagai komunikasi UART dan 6 pin yang dapat dimanfaatkan sebagai *output* PWM serta 6 pin *input* analog, 16 MHz osilator kristal, koneksi USB, ICSP *header* dan tombol reset. Pin UART juga terkoneksi pada *converter USB-to-TTL* sehingga data serial dapat diproses oleh komputer melalui *interface* USB. Untuk *power supply*, dapat digunakan kabel *mini-USB* maupun baterai yang dikoneksikan ke pin Vin. Tegangan yang dapat digunakan sebagai *supply* yaitu 7-12 V. Gambar 2.15 merupakan bentuk *development board* Arduino Nano



Gambar 2.15 Arduino Nano

Bentuk yang kecil merupakan satu nilai positif yang menjadi pertimbangan dalam penggunaannya, serta bentuknya yang dapat dengan mudah digabungkan dengan sistem yang lebih besar dengan memanfaatkan pinheader. Dengan spesifikasi *hardware* (tabel 2.1) yang identik dengan Arduino UNO, Arduino Nano memiliki nilai tambah dengan memanfaatkan bentuknya yang lebih kecil dan kompak.

Tabel 2.1 Spesifikasi Arduino Nano [7]

Mikrokontroller	ATmega328
Arsitektur	AVR (8 bit)
Tegangan Operasi	5V
Memori Flash	32 KB dengan 2 KB untuk bootloader
SRAM	2 KB
Kecepatan Clock	16 MHz
Jumlah pin Analog *Input*	8
EEPROM	1 KB
Arus DC per pin I/O	40 mA
Tegangan *Input*	7-12 V
Jumlah pin I/O Digital	22 (6 diantaranya dapat dioptimasi untuk PWM)
Jumlah pin PWM	6
Konsumsi Daya	19 mA
Ukuran PCB	18 × 45 mm
Berat	7 g

2.10 Quartus Prime

Intel Quartus Prime adalah *programmable logic device design software* yang diproduksi oleh Intel. Sebelum Altera diakuisisi oleh Intel, *software* ini bernama Altera Quartus. Quartus Prime memungkinkan analisis dan sintesis desain HDL, yang memungkinkan developer untuk menyusun dan mensimulasikan desain yang telah dibuat serta mengatur agar desain tersebut dapat diaplikasikan dan diprogram pada divais yang tersedia. Quartus menyediakan implementasi VHDL dan Verilog untuk *hardware description*, penyusunan rangkaian logika secara manual dengan *interface* visual serta simulasi *waveform*.

Berikut salah satu fitur yang dimiliki oleh Quartus Prime:

a. SOPC (*System on a Programmable Chip*) Builder, sebuah fitur pada Quartus Prime yang membuat interkoneksi antar *logic* dan membuat *testbench* untuk verifikasi fungsional dari interkoneksi yang dibuat. Sistem SOPC memungkinkan dibuatnya komputer pada sistem prosesor FPGA. Sistem SOPC terdapat pada hampir seluruh sistem

FPGA intel.

b. Qsys, fitur integrasi sistem yang digunakan untuk mengakses fitur-fitur yang ada pada board seperti ADC, *character* LCD, *Audio processing*, dll. Fitur ini merupakan pengembangan dari SOPC Builder.

2.11 *Liquid Crystal Display*

Liquid Crystal Display (LCD) merupakan suatu jenis media tampilan yang memanfaatkan kristal cair sebagai komponen penampil utama. LCD pada sistem ini digunakan sebagai penampil data berupa huruf, angka dan karakter-karakter grafik lainnya. LCD pada umumnya dijual sebagai modul siap pakai seperti pada gambar 2.16.

LCD memiliki beberapa jenis *register* dan beberapa kelompok pin, yaitu pin data, kontrol catu daya dan pengatur kontras dan kecerahan. LCD merupakan perangkat *display* yang umum digunakan bersama *microcontroller*. Pada aplikasinya, LCD banyak digunakan untuk menampilkan berbagai macam informasi, misalnya jam, pembacaan sensor maupun sebagai *user interface*. Beberapa register yang terdapat pada LCD diantaranya:

- IR (*Instruction Register)*

Digunakan untuk menentukan fungsi yang harus dikerjakan oleh LCD serta pengalamatan DDRAM atau CGRAM

- DR (*Data Register*)

Digunakan sebagai pin pengiriman data instruksi maupun karakter, dikirimkan setelah IR diinisiasi lebih awal.

- BF (*Busy Flag*)

Busy Flag dikontrol oleh pin R/W, register ini menentukan apakah LCD dapat dikontrol dan dimasuki *input* karakter atau tidak. Pada konfigurasi umumnya, pin R/W di ground agar *input* dapat terus masuk ke modul LCD.

- AC (*Address Counter*)

Address counter digunakan untuk menunjuk alamat pada RAM LCD untuk selanjutnya dibaca atau ditulis. AC otomatis bergeser ke alamat selanjutnya ketika operasi pembacaan atau penulisan selesai.

- DDRAM (*Display Data Random Access Memory*)

Digunakan sebagai penyimpan data karakter yang sedang ditampilkan. Ukuran DDRAM bervariasi mengikuti ukuran LCD.

- CGROM (*Character Generator Read Only Memory*)

Pada LCD terdapat ROM untuk menyimpan karakter-karakter ASCII

(*American Standart Code for Information Interchange*) sehingga cukup memasukkan kode ASCII untuk menampilkannya.

- CGRAM (*Character Generator Random Access Memory*)

Memory yang digunakan untuk menyimpan karakter-karakter *custom* yang dibuat oleh user.

- *Cursor Blink Control Circuit*

Merupakan rangkaian yang menghasilkan tampilan kursor dan kondisi *blink*.

Gambar 2.16 *Liquid Crystal Display* 16x2

Tabel 2.1 Konfigurasi Pin LCD

No. Pin	Simbol	Fungsi
1	VSS	*Ground*
2	VDD	*Power Supply*
3	V0	*Power Supply* untuk LCD
4	RS	*Selector* untuk *Display Data* (“H”) atau *Instruction* (“L”)
5	R/W	*Read or Write Select Signal*
6	E	*Read/Write Enable Signal*
7	DB0	*Display Data Signal*
9	…	
10	DB7	

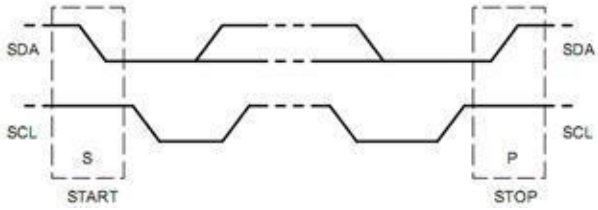
11	LED – (K)	Terminal katoda untuk *Backlight LED*
12	LED + (A)	Terminal anoda untuk *Backlight LED*

2.12 I²C (*Inter Integrated Circuit*)

Inter Integrated Circuit atau sering disebut I2C adalah standar komunikasi serial dua arah menggunakan dua saluran yang didesain khusus untuk mengirim maupun menerima data. Sistem I2C terdiri dari saluran SCL (*Serial Clock*) dan SDA (*Serial Data*) yang membawa informasi data antara I2C dengan pengontrolnya. Contoh modul I2C yang beredar di pasaran adalah modul I2C LCD module (Gambar 2.17).

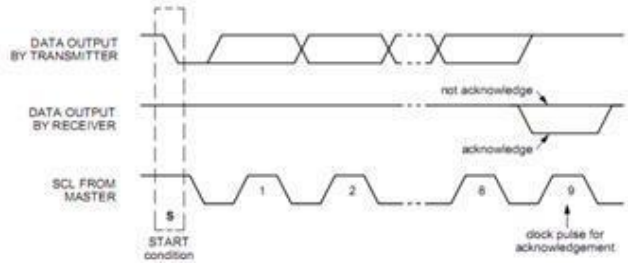
Gambar 2.17 Comtoh Modul I²C (I²C LCD *Module*)

Piranti yang dihubungkan dengan sistem I2C Bus dapat dioperasikan sebagai *Master* dan *Slave*. *Master* adalah piranti yang memulai *Transfer* data pada I2C Bus dengan membentuk sinyal *Start*, mengakhiri *Transfer* data dengan membentuk sinyal *Stop*, dan membangkitkan sinyal clock. *Slave* adalah piranti yang dialamati *Master*. Sinyal *Start* merupakan sinyal untuk memulai semua perintah, didefinisikan sebagai perubahan tegangan SDA dari "1" menjadi "0" pada saat SCL "1". Sinyal *Stop* merupakan sinyal untuk mengakhiri semua perintah, didefinisikan sebagai perubahan tegangan SDA dari "0" menjadi "1" pada saat SCL "1". Kondisi sinyal *Start* dan sinyal *Stop* seperti tampak pada Gambar 2.18.



Gambar 2.18 Kondisi Sinyal *Start* dan *Stop*

Sinyal dasar yang lain dalam I2C Bus adalah sinyal *acknowledge* yang disimbolkan dengan ACK Setelah *Transfer* data oleh *Master* berhasil diterima *slave*, *slave* akan menjawabnya dengan mengirim sinyal *acknowledge*, yaitu dengan membuat SDA menjadi "0" selama siklus *clock* ke 9. Ini menunjukkan bahwa *Slave* telah menerima 8 bit data dari *Master*. Kondisi sinyal *acknowledge* seperti tampak pada Gambar 2.19.

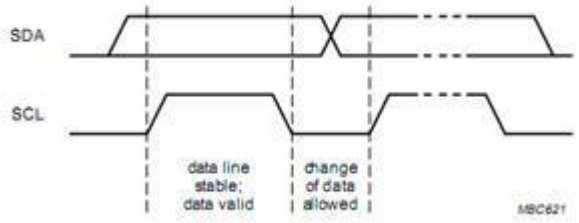


Gambar 2.19 Sinyal ACK dan NACK

Dalam melakukan *Transfer* data pada I2C Bus (Gambar 2.20), kita harus mengikuti tata cara yang telah ditetapkan yaitu:

- *Transfer* data hanya dapat dilakukan ketikan Bus tidak dalam keadaan sibuk.
- Selama proses *Transfer* data, keadaan data pada SDA harus stabil selama SCL dalam keadan tinggi. Keadaan perubahan "1" atau "0" pada SDA hanya dapat dilakukan selama SCL dalam keadaan rendah. Jika terjadi perubahan keadaan SDA pada saat

SCL dalam keadaan tinggi, maka perubahan itu dianggap sebagai sinyal *Start* atau sinyal *Stop*.



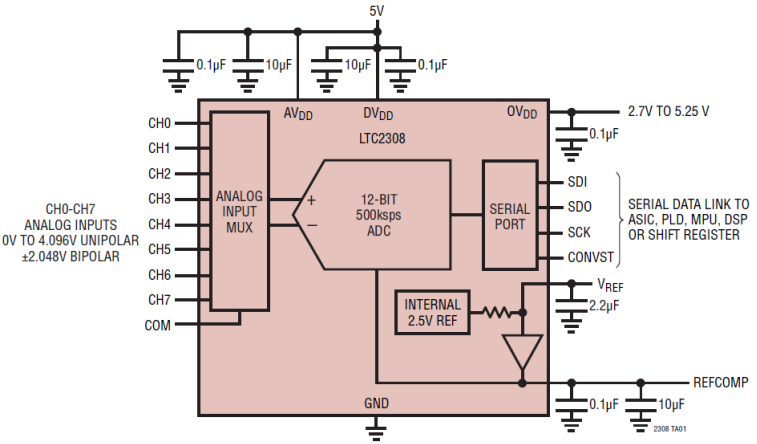
Gambar 2.20 *Transfer Bit pada I^2C Bus*

2.13 *Analog-to-Digital Converter*

Dalam bidang elektronik, *analog-to-digital converter* merupakan sistem yang mengubah sinyal analog seperti sinyal suara yang ditangkap oleh mikrofon atau cahaya yang masuk melalui kamera digital menjadi sinyal digital. ADC juga dapat dijadikan sebagai alat ukur terisolasi seperti yang digunakan pada peralatan elektronik yang mengubah *input* tegangan atau arus analog menjadi angka digital yang merepresentasikan besar tegangan atau arus yang bersangkutan.

ADC mengkonversi sinyal analog dengan waktu dan amplitude kontinyu menjadi sinyal digital dengan waktu dan amplitude diskrit. Proses konversi ini melibatkan proses kuantisasi dari sinyal *input* sehingga sejumlah kecil error atau noise pasti akan terjadi. Selain dari itu, proses konversi tidak dilakukan secara kontinyu namun secara periodik mengambil sampling dari *input*, sehingga membatasi *bandwidth* yang diijinkan oleh sinyal *input*.

ADC pada DE-10 Nano menggunakan IC LTC2308 yang menggunakan sistem *Successive Approximation Registers*. LTC2308 dilengkapi dengan 8-to-1 mltiplexer sehingga dapat digunakan 8 jalur *input* analog. Tegangan referensi internal sebesar 2.5V namun tetap disediakan pin external untuk tegangan referensi yang dapat ditentukan oleh user sendiri. Bentuk penggunaan IC ADC ini secara umum digambarkan pada gambar 2.21.



Gambar 2.21 Aplikasi Umum IC LTC2308

LTC2308 menggunakan sistem komunikasi SPI untuk mengirimkan outputnya. Sistem komunikasi SPI dengan ADC ditangani oleh Quartus sepenuhnya melalui *Intelectual Properties* yang disediakan pada IP catalog.

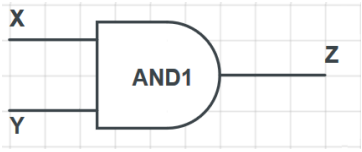
Salah satu faktor yang mempengaruhi kualitas sinyal yang di *sampling* ADC ada pada resolusi dari ADC sendiri. Resolusi dari tiap bit ADC dapat dihitung dengan membagi tegangan referensi dengan 2^n-1, dimana nilai n merupakan nilai resolusi dari ADC. Semakin tinggi resolusi ADC maka hasil sampling akan semakin mendekati sinyal aslinya.

2.14 Rangkaian Logika

Boolean logic, beroperasi menggunakan sistem bilangan biner, di mana hanya terdapat 2 kemungkinan yaitu “1” dan “0” pada setiap digitnya. Fungsi dasar dari gerbang logika merupakan operasi dari *input-input* yang diberikan, jumlah *input* dapat berupa *input* tunggal maupun jamak. Tergantung gerbang apa yang digunakan, serta berapa *input* yang terdapat pada gerbang yang bersangkutan. Dalam aplikasi di lapangan, sebagian besar gerbang-gerbang logika disusun oleh *field-effect transistor* (FET), terutama *metal-oxide-semiconductor field-effect transistor* (MOSFET). Beberapa contoh gerbang logika yang digunakan untuk membentuk rangkaian logika diantaranya:

a. Gerbang AND

Simbol Gerbang AND

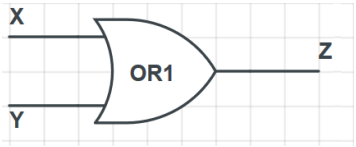


Tabel Kebenaran Gerbang AND

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

b. Gerbang OR

Simbol Gerbang OR

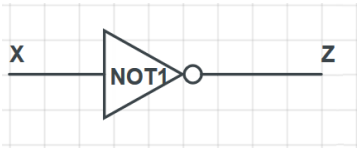


Tabel Kebenaran Gerbang OR

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

c. Gerbang NOT

Simbol Gerbang NOT

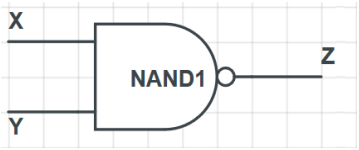


Tabel Kebenaran Gerbang NOT

X	Z
0	1
1	0

d. Gerbang NAND

Simbol Gerbang NAND

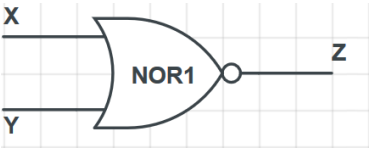


Tabel Kebenaran Gerbang NAND

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

e. Gerbang NOR

Simbol Gerbang NOR

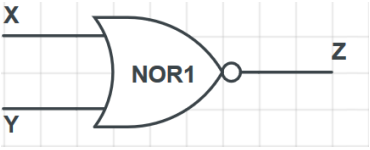


Tabel Kebenaran Gerbang NOR

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

f. Gerbang X-OR (Exclusive OR)

Simbol Gerbang X-OR

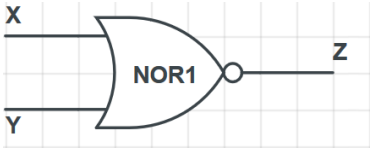


Tabel Kebenaran Gerbang X-OR

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

g. Gerbang X-NOR (Exclusive NOR)

Simbol Gerbang X-NOR **Tabel Kebenaran Gerbang X-NOR**

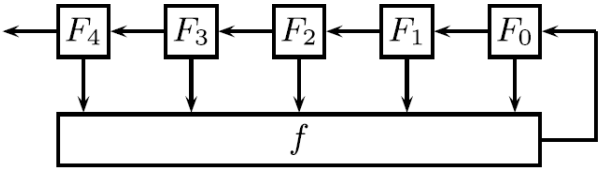


X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

Salah satu jenis gerbang logika sederhana yang menggunakan transistor bipolar (BJT) adalah *resistor-transistor logic* (RTL). RTL merupakan satu bentuk *logic* yang diimplementasikan pada IC generasi awal. Seiring berkembangnya waktu, RTL diganti dengan *transistor-transistor logic* (TTL). Namun, seiring berkembangnya desain IC yang membutuhkan kerapatan yang lebih dan tuntutan untuk mengurangi konsumsi daya, BJT digantikan oleh *field-effect transistor* (MOSFET). Dengan ditemukan kedua jenis MOSFET, konfigurasi CMOS (*Complementary* MOSFET) digunakan dikarenakan konsumsi daya dan kecepatan yang dinilai lebih baik.

2.15 *Linear Feedback Shift Register* (LFSR)

Feedback Shift Register merupakan satu alat yang cukup berguna sebagai generator *pseudo-random number* dan kriptografi. Dalam *feedback shift register*, dalam setiap *time step*, F_i mengambil nilai F_{i-1} untuk i > 0 dan F_0 diupdate berdasarkan fungsi feedback tertentu[8]. Secara sederhana, fungsi tersebut diilustrasikan seperti pada gambar 2.22



Gambar 2.22 *Feedback Shift Register* **[8]**

Linear feedback shift register merupakan *shift register* yang *input* bitnya merupakan fungsi linear dari kondisi sebelumnya. Fungsi linear yang biasa digunakan sebagai feedback adalah exclusive-or (XOR). Sehingga, LFSR seringkali berupa *shift register* yang dikontrol XOR dari beberapa bit dari keseluruhan nilai *shift register*.

Nilai awal dari LFSR disebut sebagai seed, dan karena operasi dari LFSR sudah ditentukan maka data yang dihasilkan register akan ditentukan oleh nilai *state* saat ini dan/atau *state* sebelumnya. LFSR memiliki batasan *state*, oleh karena itu pada titik tertentu nilai register akan terus berulang. Namun, dengan penggunaan fungsi *feedback* yang tepat dapat memberikan hasil yang terkesan acak dan memiliki siklus yang perulangan yang sangat panjang.

2.16 *Cross Correlation*

Setelah dilakukan konversi, perlu dilakukan penyimpanan nilai-nilai stokastik untuk selanjutnya dioperasikan pada proses *cross-correlation*. Seluruh data hasil konversi dari dua channel disimpan dalam 128 *slot* array. Tujuan penyimpanan ini agar didapatkan aliran data yang konsisten pada operasi *cross-correlation*. Metode ini digunakan untuk mencari korelasi (tingkat kesamaan) dari kedua sinyal, pada kasus ini dicari pergeseran fasa dengan mencocokkan kedua sinyal yang digeser berdasarkan waktu sehingga didapatkan jeda atau perbedaan waktu sampai sumber suara menuju kedua mikrofon.

$$Corr(g,h)_j(t) \equiv \sum_{k=0}^{N-1} g_{j+k} h_k \tag{2.4}$$

Fungsi *cross-correlation* didefinisikan pada persamaan 2.4, fungsi korelasi *Corr(g,h)(t)* akan memiliki nilai masing-masing seiring digesernya fungsi *g(t)* digeser seiring waktu melalui persamaan h(t). Nilai-nilai tersebut akan dicari nilai tertingginya, sehingga akan didapatkan nilai *delay* dari kedua sinyal menggunakan *time delay analysis* seperti pada persamaan 2.5.

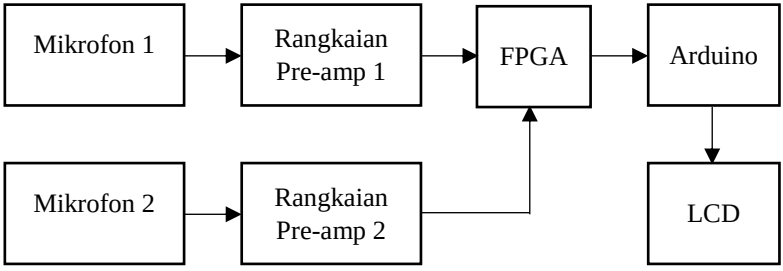
$$\tau_{delay} = \arg\max((g \star h)(t)) \tag{2.5}$$

Halaman ini sengaja dikosongkan

BAB III
PERANCANGAN SISTEM

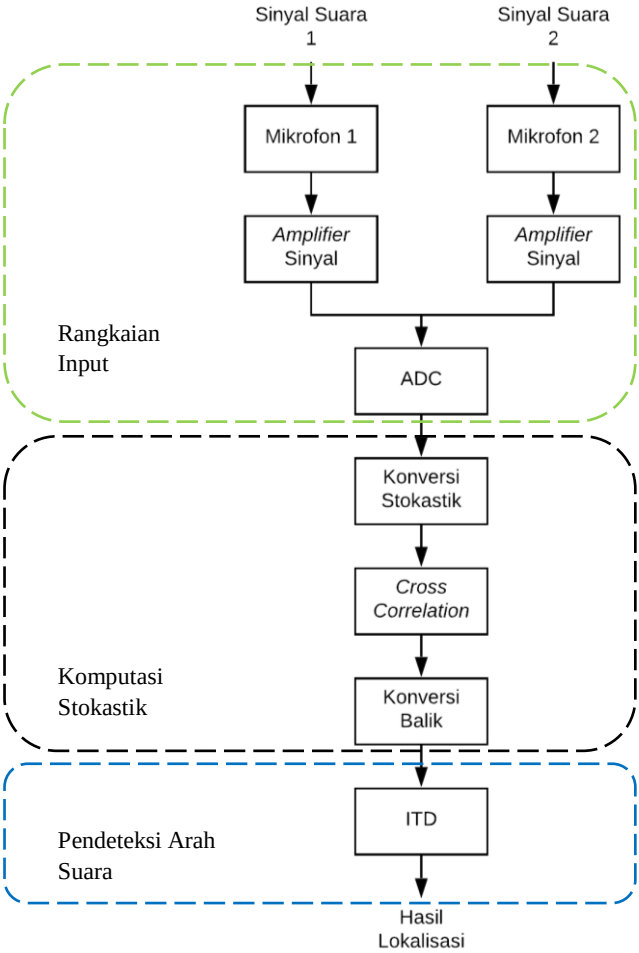
Pada bab ini dilakukan pembahasan mengenai desain keseluruhan dari sistem yang dibuat. Sistem lokalisasi suara menggunakan FPGA berbasis komputasi stokastik akan digunakan untuk mendapatkan sumber arah suara yang selanjutnya akan ditampilkan pada LCD dengan memanfaatkan sistem komunikasi I^2C yang ada pada Arduino. Tugas akhir ini akan lebih fokus pada perancangan *software*, namun desain *hardware* juga akan dibahas karena akan terdapat beberapa rangkaian tambahan beserta modifikasi pada FPGA agar sistem dapat bekerja. Diagram hardware sistem secara keseluruhan dapat dilihat pada gambar 3.1 dan gambar 3.2 menggambarkan diagram proses.

Dari diagram blok yang ditampilkan, dapat dilihat sistem yang digunakan cukup sederhana dan linier. Sinyal suara ditangkap oleh kedua microphone, selanjutnya dikuatkan oleh rangkaian *pre-amplifier* sehingga dapat dibaca dengan baik oleh ADC yang terintegrasi dengan FPGA. Pada tahap selanjutnya, dua vektor data akan dikonversi dan diproses pada FPGA menggunakan sistem komputasi stokastik dan dikeluarkan sebagai sinyal *parallel* 8 bit yang akan ditampilkan pada LCD dengan memanfaatkan sistem komunikasi I^2C yang ada pada Arduino nano. Sistem ini memanfaatkan *delay* antara kedua sinyal suara yang ditangkap oleh mikrofon. Ketika jarak sumber suara lebih dekat ke mikrofon 1, maka mikrofon 1 akan menerima suara lebih cepat dari mikrofon 2, begitu pula sebaliknya. Setelah beda waktu didapatkan, maka hasil arah sumber suara juga dapat diperoleh.



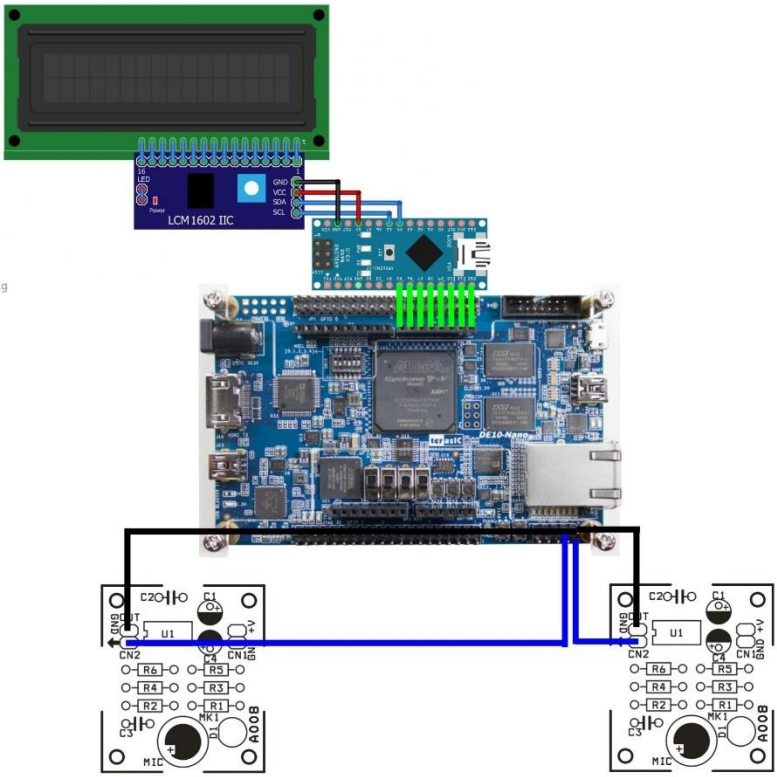
Gambar 3.1 Diagram Hardware Sistem Secara Keseluruhan

Sistem menggunakan adaptor bawaan FPGA dan kabel usb yang juga berfungsi sebagai power supply untuk Arduino. Rangkaian pre-amp juga didesain sedemikian rupa sehingga tidak memerlukan dual supply dan dapat memanfaatkan output tegangan 5v dari board FPGA.



Gambar 3.2 Diagram Proses Sistem Keseluruhan

Pada sistem ini, digunakan 3 set *hardware* yaitu mikrofon dan *pre-amplifier*, DE-10 Nano FPGA *Development Board*, serta Arduino, perancangan *hardware* meliputi perancangan mikrofon dan *pre-amplifier*, serta hubungan antara FPGA, Arduino, mikrofon dan LCD.



Gambar 3.3 Interkoneksi Antar Komponen

Sistem ini memanfaatkan ADC bawaan yang telah disediakan oleh DE10-Nano. Channel ADC yang digunakan pada sistem ini adalah CH0 dan CH1 yang terhubung pada output pre-amplifier. *Power supply* untuk rangkaian *pre-amplifier* serta *grounding reference* untuk board Arduino juga disediakan oleh DE10-Nano. Output dari FPGA akan dikirim ke Arduino melalui 8 pin Arduino header. Selain itu akan digunakan juga 4 pin dari GPIO untuk memfasilitasi pengiriman data dari

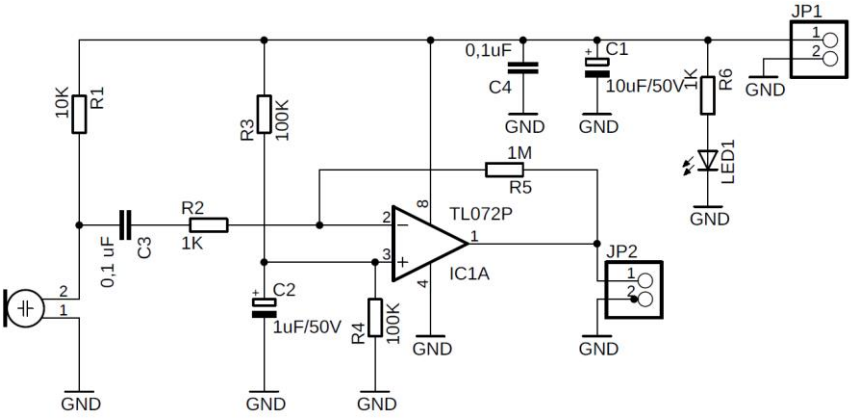
segi *timing* dan pemilahan data, mengingat perbedaan clock dan data yang dikirimkan oleh FPGA lebih dari satu jenis. Penempatan pin digambarkan oleh gambar 3.3.

Setelah dilakukan proses pada FPGA, hasil dikirim secara parallel melalui 8 pin yang telah digambarkan pada gambar 3.3. Selanjutnya hasil proses dari Arduino akan ditampilkan pada LCD dengan memanfaatkan modul I^2C LCD.

3.1 Rangkaian Input

3.1.1 Mikrofon dan Pre-Amplifier

Perangkat pre-amplifier merupakan perangkat penguat sinyal yang didesain untuk memperkuat sinyal dengan tegangan dan arus rendah, seperti yang terdapat pada mikrofon ataupun sensor-sensor analog pada umumnya. Pre-amplifier yang digunakan akan berbasis IC TL072 yang merupakan IC *dual op-amp*. Sinyal yang ditangkap oleh mikrofon pada umumnya berkisar pada beberapa mV yang akan sulit ditangkap dengan baik oleh ADC. Mikrofon yang digunakan merupakan mikrofon condenser elektret yang sering ditemukan di pasaran. *Pre-amplifier* yang digunakan pada sistem (gambar 3.4) memanfaatkan tegangan suplai 5V dari FPGA.



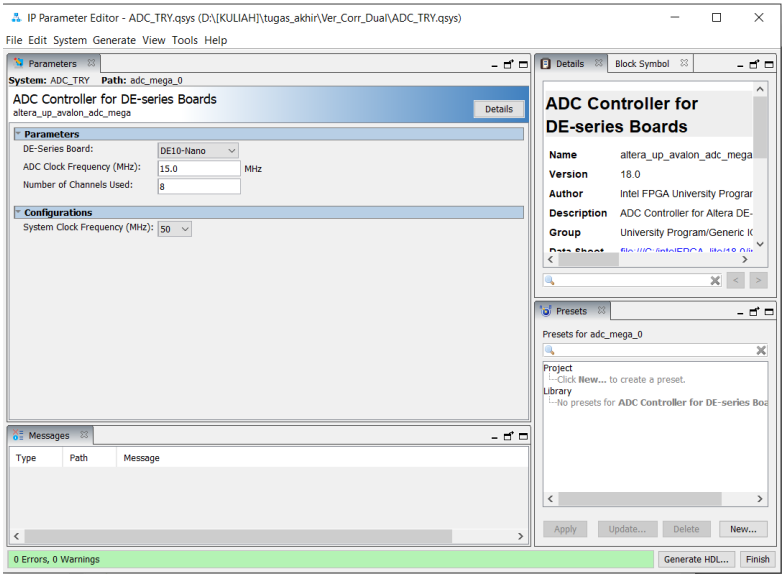
Gambar 3.4 Rangkaian Pre-Amplifier

LM358 merupakan rangkaian terintegrasi yang terdiri atas dua op-amp dengan gain tinggi, dengan frekuensi telah terkompensasi secara

internal yang didesain agar dapat beroperasi dengan *power supply* tunggal. Aplikasi dari IC terkait diantaranya penguat transduser, penguat tegangan dc dan rangkaian op-amp konvensional. TL072 dipilih karena dukungan *supply* tunggal sehingga tidak diperlukan rangkaian *center-tap* untuk mengganti *supply* ganda. Op-amp juga memungkinkan rangkaian pre-amplifier dibuat lebih sederhana.

3.1.2 Perancangan ADC

Sinyal hasil output *pre-amplifier* akan diterima oleh ADC pada *board* DE-10 Nano. Proses ini memanfaatkan IP Platform Designer yang telah disediakan oleh Quartus Prime, sehingga data output dari ADC dapat diakses dengan lebih mudah. IP yang digunakan untuk menggunakan ADC adalah *ADC Controller for DE-series Boards* (gambar 3.5) pada *IP Library* yang terdapat pada *software* Quartus Prime.



Gambar 3.5 IP Parameter Editor untuk ADC

ADC *clock frequency* dapat ditentukan antara 0,01 MHz hingga 20 MHz. *Sampling rate* dari ADC bergantung pada *clock frequency* yang ditentukan pada *parameter editor*. 12 bit data output dari ADC dapat

diterima setiap 16 clock untuk setiap channel. sehingga dengan menggunakan konfigurasi 15 MHz *clock frequency* dan 2 channel, frekuensi sampling ADC adalah,

$$f_{sampling} = \frac{15\,MHz}{16 \times 2} = 0{,}46875 \text{ MHz} = 468{,}75\text{KHz}$$

Sistem yang dibuat memanfaatkan buffer update setiap 10*256 clock, sehingga data yang dikonversi menjadi bilangan stokastik diperbaharui dengan frekuensi

$$f_{update} = \frac{50\,MHz}{5 \times 256} = 39{,}06\,KHz$$

Mengacu bahwa $f_{update} < f_{sampling}$, sehingga data yang didapatkan setiap update berbeda dari sebelumnya. Maka dapat disimpulkan bahwa data yang digunakan sebagai *input* merupakan data yang valid.

Untuk mengukur nilai yang diukur, mengacu pada sistem ADC 12 bit yang ada pada board, resolusi yang didapatkan untuk setiap sample yang diambil adalah,

$$resolusi = \frac{V_{ref}}{2^{12}-1} = \frac{4000\,mV}{4095} = 0{,}9768\,mV/bit$$

Namun dikarenakan komputasi stokastik yang dirancang hanya sampai 8 bit mengingat bahwa sistem ini memakan cukup banyak *resource* serta Arduino sebagai perantara hanya diberikan 8 pin untuk pengiriman data maka data yang diambil dari 12 bit ADC hanya sebanyak 8 bit MSB.

IP memiliki konsep kerja yang mirip dengan module pada Verilog, sehingga diperlukan *command line* tertentu yang mendefinisikan *input* dan output yang akan dioperasikan pada IP tersebut sebagai inisialisasi ADC. *Command line* yang digunakan oleh IP *ADC Controller for DE-series Boards* adalah

```
ADC_TRY(.CLOCK (FPGA_CLK1_50),
        .RESET  (!KEY[0]),
        .CH0        (outa),
        .CH1        (outb),
        .ADC_SCLK           (ADC_SCK),
        .ADC_CS_N           (ADC_CONVST),
```

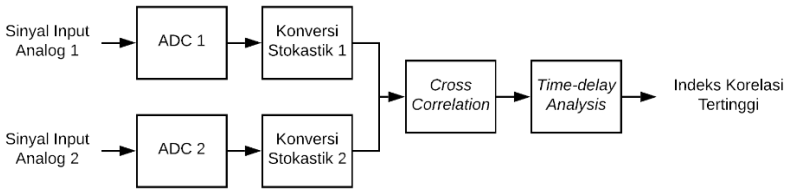
```
.ADC_DOUT       (ADC_SDO),
.ADC_DIN        (ADC_SDI));
```

IP *ADC Controller* dapat dinamai sesuai keinginan melalui *parameter editor*. Di dalam tanda kurung, didefinisikan *input* dan output yang digunakan dalam operasi pada IP yang bersangkutan.

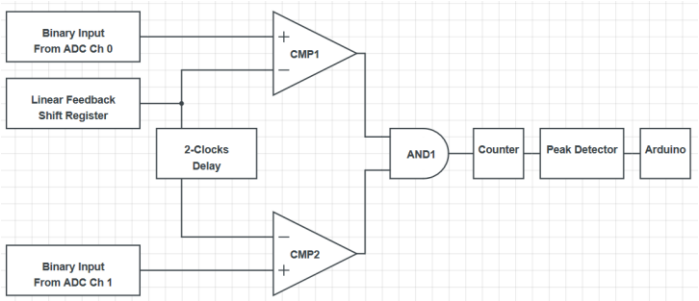
3.2 Komputasi Stokastik

Setelah mengalami penguatan pada rangkaian pre-amplifier, sinyal audio akan diproses pada FPGA DE-10 Nano menggunakan komputasi stokastik sehingga didapatkan sumber arah suara. Bahasa yang digunakan adalah bahasa Verilog HDL menggunakan *software* Quartus Prime 18.0. Setelah itu data akan dikirim secara parallel ke Arduino Nano untuk selanjutnya dikirim ke PC menggunakan komunikasi Serial. Data yang dikirim akan ditampilkan melalui terminal serial pada PC.

Sistem komputasi stokastik dimulai setelah sampling sinyal audio memanfaatkan *analog-to-digital converter* yang terdapat pada *board* De-10 Nano. Sinyal output dari ADC akan langsung disimpan dalam dua *array*, *Array* yang telah terbentuk akan dikonversi menjadi bilangan stokastik 256 bit dengan memanfaatkan komparator dan *Linear Feedback Shift Register* lalu diproses sehingga dapat ditemukan delay antara kedua *input* yang dapat dikirim ke Arduino untuk ditemukan arah suaranya untuk selanjutnya ditampilkan di LCD. Secara sederhana, *behavioural* sistem FPGA digambarkan oleh gambar 3.6 Estimasi bentuk rangkaian internal FPGA digambarkan oleh gambar 3.7.



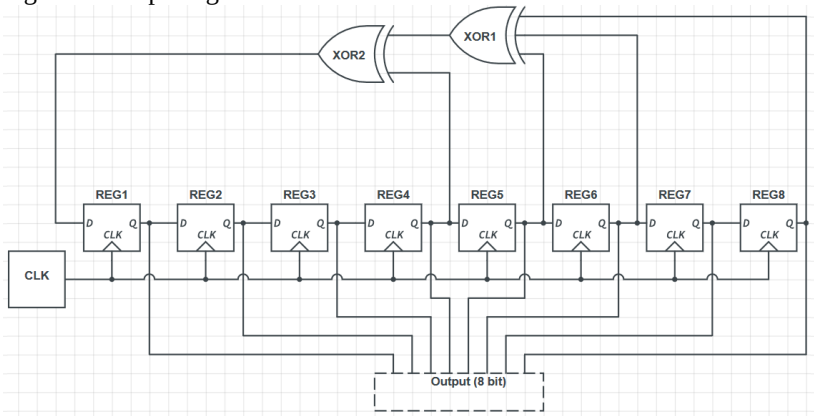
Gambar 3.6 Diagram Sistem Rangkaian FPGA



Gambar 3.7 Estimasi Rangkaian Internal FPGA

3.2.1 Perancangan *Linear Feedback Shift Register*

Linear Feedback Shift Register digunakan sebagai *random number input* pada komparator yang digunakan pada proses konversi bilangan stokastik. Bilangan biner *input* yang akan dikonversi ditentukan sejumlah 8 bit, sehingga dibutuhkan bilangan biner acak dengan jumlah bit yang sama. Representasi bentuk rangkaian yang digunakan digambarkan pada gambar 3.8.



Gambar 3.8 Rangkaian LFSR 8 bit

Rangkaian LFSR tersebut akan mengeluarkan angka-angka acak sejumlah 8 bit yang dapat digunakan sebagai *input* dari komparator. LFSR memiliki basis operasi sebagai *shift register* sehingga nilai bit dari setiap output akan digeser seiring dengan *clock* yang digunakan pada LFSR.

Rangkaian XNOR digunakan sebagai penentu feedback yang akan dimasukkan kembali ke *shift register*.

Menggunakan konfigurasi polynomial F= $x^8+x^6+x^5+x^4+1$, jumlah kemungkinan nilai LFSR yang didapatkan sejumlah 255. Nilai ini dinilai cukup untuk membantu konversi dari bilangan biner 8 bit menjadi bilangan stokastik 256 bit.

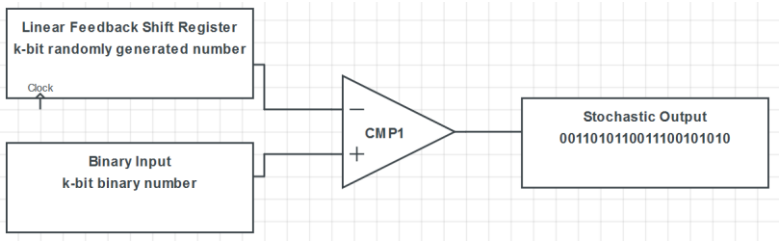
Pada sistem ini, digunakan modul Verilog untuk mempermudah penulisan pada modul utama. Modul yang digunakan menyediakan cukup banyak pilihan sesuai dengan bit yang dibutuhkan.

```
LFSR #(.NUM_BITS(c_NUM_BITS)) LFSR_inst
      (.i_Clk(clk),
       .i_Enable(1'b1),
       .i_Seed_DV(1'b0),
       .i_Seed_Data({c_NUM_BITS{1'b0}}), // Replication
       .o_LFSR_Data(lfsr_out),
       .o_LFSR_Done()
       );
```

Berikut merupakan *command line* yang digunakan untuk menginisiasi modul LFSR.

3.2.2 Konversi Bilangan Stokastik

Setelah dilakukan sampling dan *random number generation*, selanjutnya dilakukan proses konversi ke bilangan stokastik. Konversi dilakukan menggunakan komparator seperti pada gambar 3.9. Bilangan ter-sampling akan dibandingkan dengan bilangan random yang dihasilkan oleh LFSR. Komparator yang digunakan diharuskan memiliki *input* 8 bit untuk masing-masing *input*. Setelah komparasi, akan dilakukan bit-shift sehingga nilai awal tetap disimpan dan nilai baru akan dikomparasikan. Proses bit-shift dan komparasi akan terus dilakukan untuk satu nilai sampling hingga hasil output komparator mencapai 256 bit. 256 bit dipilih, untuk menyesuaikan dengan besar *input* yang digunakan. Dengan menggunakan 8 bit *input* biner, untuk menyamai akurasinya diperlukan 2^n bit bilangan stokastik.



Gambar 3.9 Generator Bilangan Stokastik

Setelah dilakukan konversi untuk satu baris bilangan stokastik, titik sampling akan digeser hingga didapatkan 127 titik sampling yang telah dikonversi untuk selanjutnya dioperasikan untuk menemukan titik korelasi tertinggi.

Komparator 8 bit antara A dan B dapat dibangun dengan membandingkan MSB dan turun membandingkan LSB sehingga dapat dibuat persyaratan-persyaratan berupa:

a. Jika A7=1 dan B7=0 maka A lebih besar dari B (A>B) atau
b. Jika A7 dan B7 sama, dan jika A6=1 dan B6=0 maka A lebih besar dari B (A>B)
c. Jika A7 dan B7 sama, A6 dan B6 sama, dan jika A5=1 dan B5=0 maka A lebih besar dari B (A>B)
d. Jika A7 dan B7 sama, A6 dan B6 sama, A5 dan B5 sama, dan jika A4=1 dan B4=0 A lebih besar dari B (A>B)
e. Jika A7 dan B7 sama, A6 dan B6 sama, A5 dan B5 sama, A4 dan B4 sama, dan jika A3=1 dan B3=0 A lebih besar dari B (A>B)
f. Jika A7 dan B7 sama, A6 dan B6 sama, A5 dan B5 sama, A4 dan B4 sama, A3 dan B3 sama dan jika A2=1 dan B2=0 A lebih besar dari B (A>B)
g. Jika A7 dan B7 sama, A6 dan B6 sama, A5 dan B5 sama, A4 dan B4 sama, A3 dan B3 sama A2 dan B2 sama dan jika A1=1 dan B1=0 A lebih besar dari B (A>B)
h. Jika A7 dan B7 sama, A6 dan B6 sama, A5 dan B5 sama, A4 dan B4 sama, A3 dan B3 sama A2 dan B2 sama, A1 dan B1 sama dan jika A0=1 dan B0=0 A lebih besar dari B (A>B)

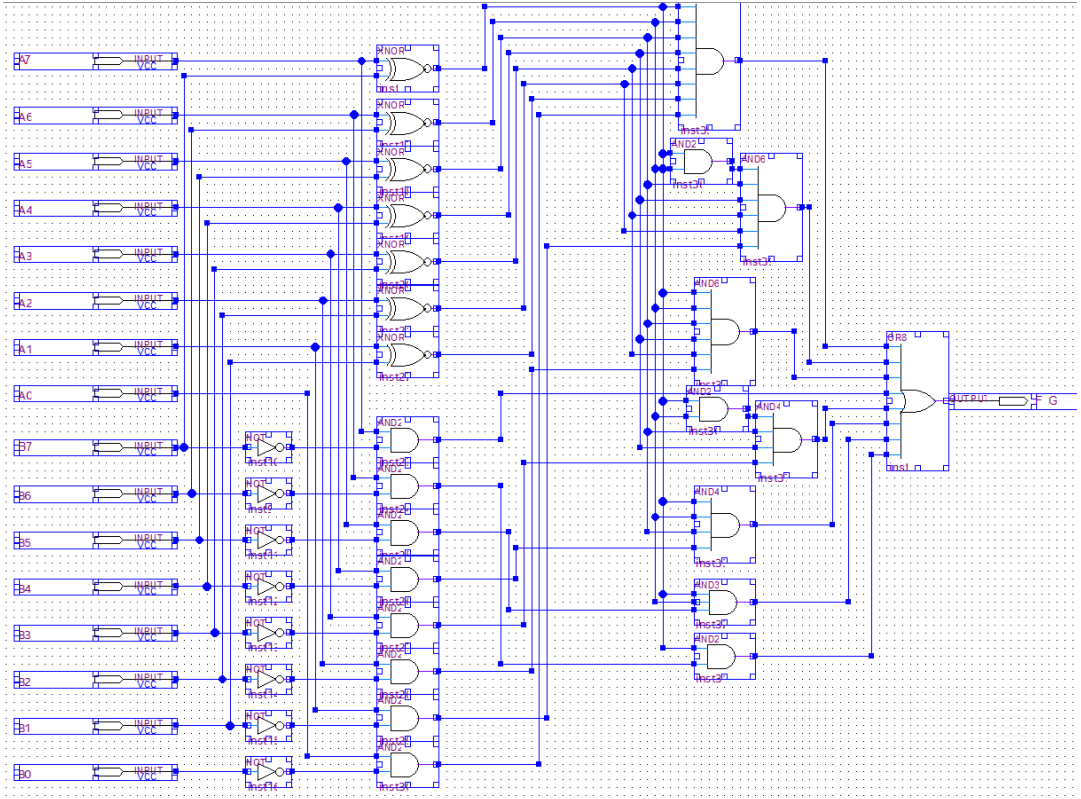
Dengan mempertimbangkan persyaratan-persyaratan tersebut, pernyataan komparator A>B dapat dirumuskan ekpresi logika sebagai berikut:

$$F_G = A7\,\overline{B7}$$
$$+(A7\ XNOR\ B7)\ A6\,\overline{B6}$$
$$+(A7\ XNOR\ B7)\ (A6\ XNOR\ B6)\ A5\,\overline{B5}$$
$$+\ (A7\ XNOR\ B7)\ (A6\ XNOR\ B6)\ (A5\ XNOR\ B5)\ A4\,\overline{B4}$$
$$+\ (A7\ XNOR\ B7)\ (A6\ XNOR\ B6)\ (A5\ XNOR\ B5)\ (A4\ XNOR\ B4)\ A4\,\overline{B4}$$
$$+\ (A7\ XNOR\ B7)\ (A6\ XNOR\ B6)\ (A5\ XNOR\ B5)\ (A4\ XNOR\ B4)\ (A3\ XNOR\ B3)\ A2\,\overline{B2}$$
$$+(A7\ XNOR\ B7)\ (A6\ XNOR\ B6)\ (A5\ XNOR\ B5)\ (A4\ XNOR\ B4)\ (A3\ XNOR\ B3)\ (A2\ XNOR\ B2)\ A1\,\overline{B1}$$
$$+(A7\ XNOR\ B7)\ (A6\ XNOR\ B6)\ (A5\ XNOR\ B5)\ (A4\ XNOR\ B4)\ (A3\ XNOR\ B3)\ (A2\ XNOR\ B2)\ (A1\ XNOR\ B1)\ A0\,\overline{B0}$$

Dalam Verilog HDL ekspresi logika tersebut dapat direpresentasikan dengan mudah yaitu dengan perintah

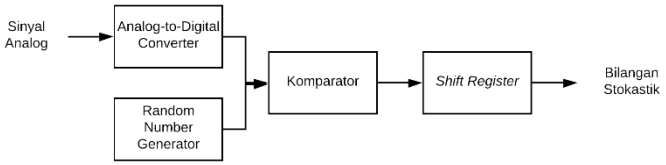
$$output <= (input\ a > input\ b)?\ 1:\ 0;$$

Lalu dengan mengacu pada parameter-parameter yang telah ditentukan, maka dapat dibuat rangkaian logika yang menggambarkan perintah tersebut, diilustrasikan pada gambar 3.10.



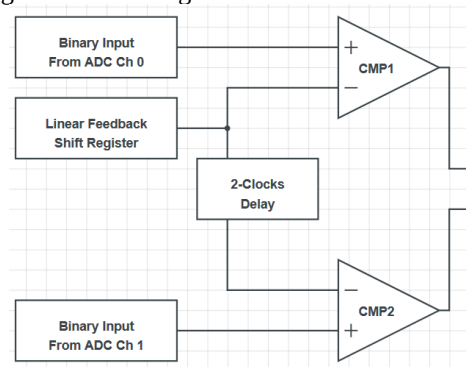
Gambar 3.10 Rangkaian Komparator 8 bit

Proses konversi terdiri dari dua langkah utama yaitu pengambilan data dari LFSR dan ADC dan komparasi antara kedua nilai. Setelah dilakukan kedua proses tersebut, komputasi dapat dilakukan.



Gambar 3.11 Diagram *Behavioural* Rangkaian Konversi Satu Baris Bilangan Stokastik 256 bit

Diagram pada gambar 3.11 menggambarkan proses konversi ke bilangan stokastik secara keseluruhan. Dimulai dari pengambilan buffer ADC di setiap clock kelipatan ke 256, komparasi nilai ADC dengan LFSR setiap clocknya dan penyimpanan bilangan stokastik 256 bit pada array sejumlah 127 setiap channelnya. Sedangkan estimasi rangkaian pada FPGA digambarkan oleh gambar 3.12

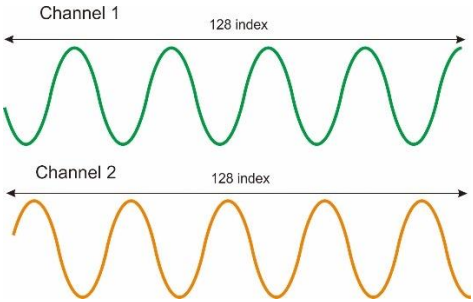


Gambar 3.12 Estimasi Rangkaian Konversi Stokastik

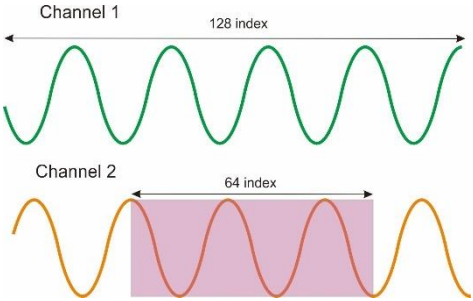
3.2.3 *Cross Correlation*

Program *cross-correlation* dalam sistem digunakan untuk mencari *delay* sampainya sinyal suara dari sumber menuju kedua mikrofon, proses ini dilakukan setelah proses penyimpanan data yang akan dijadikan *input* proses *cross-correlation*. Selanjutnya dilakukan perkalian stokastik dan dilanjutkan dengan “integrasi” stokastik, yang

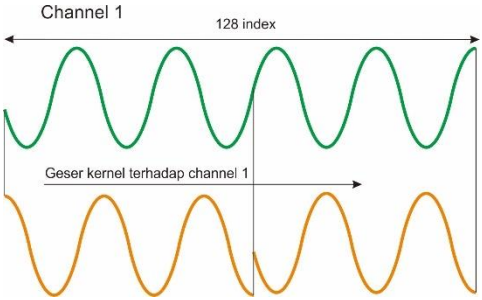
akan menghasilkan nilai-nilai korelasi antara kedua sinyal. Ilustrasi proses yang dilakukan digambarkan pada gambar 3.13, 3.14 dan 3.15.



Gambar 3.13 *Input* Gelombang 2 *Channel* untuk Proses *Cross-Correlation*



Gambar 3.14 Penentuan *Kernel* pada *Channel* 2

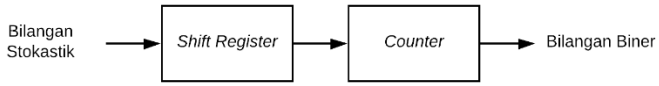


Gambar 3.15 Proses Penggeseran *Kernel* dan Pencarian Korelasi Tertinggi

Proses di atas digunakan untuk menghitung korelasi antara kedua sinyal sekaligus mencari nilai korelasi tertinggi untuk dioperasikan lebih lanjut agar dapat ditemukan nilai *delay* antara kedua gelombang.

3.2.4 Konversi Balik ke Bilangan Biner

Setelah dilakukan perhitungan *cross-correlation* dan ditemukan nilai *delay*, hasil-hasil tersebut akan dikirimkan ke Arduino. Namun, sebelum dilakukan pengiriman, bilangan stokastik akan dikembalikan menjadi bilangan biner. Proses konversi balik dari bilangan stokastik ke bilangan biner, sistem ini memanfaatkan rangkaian counter.



Gambar 3.16 Diagram *Behavioural* Rangkaian Konversi Balik

Konversi pada sistem ini akan mengembalikan bilangan stokastik 256 bit kembali menjadi 8 bit biner agar dapat lebih mudah dikirimkan ke Arduino untuk diproses lebih lanjut lalu ditampilkan ke LCD atau di-*debug* menggunakan *serial terminal* pada Arduino IDE. Bagaimana sistem ini bekerja digambarkan oleh gambar 3.16.

3.3 Lokalisasi Suara

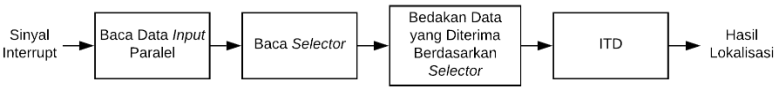
3.3.1 Pengiriman Data oleh FPGA

Setelah dilakukan perhitungan *cross-correlation* dan ditemukan nilai *delay*, hasil-hasil tersebut akan dikirimkan ke Arduino. Proses pengiriman data memanfaatkan fitur interrupt pada Arduino sebagai *trigger* dan menggunakan selektor sehingga data yang dikirimkan bisa lebih dari satu jenis. Sistematika pengiriman data dapat dilihat pada gambar 3.17.

Gambar 3.17 Diagram B*ehavioural* Rangkaian *Output* FPGA

3.3.2 Penerimaan data dari FPGA oleh Arduino

Selain pada FPGA, Arduino juga perlu diprogram agar dapat menerima data *parallel* dari FPGA. Data yang diterima oleh Arduino akan diformat sedemikian rupa agar dapat dengan mudah diamati melalui terminal serial, baik dalam berupa barisan data maupun grafik yang telah disediakan oleh Arduino IDE.



Gambar 3.18 Diagram Program Arduino untuk Menerima Data dari FPGA

Program penerima pada Arduino memiliki algoritma yang sederhana (Gambar 3.18). Arduino menerima *interrupt* dari FPGA untuk *trigger* awal, selanjutnya tipe data yang dikirimkan dapat ditentukan melalui selector. Data yang dikirimkan didapatkan dari pin-pin yang dihubungkan pada FPGA. Setelah data didapatkan secara lengkap, selanjutnya akan dikirim ke PC memanfaatkan komunikasi serial dan untuk data indeks maksimum akan dihitung kembali menggunakan rumus ITD lalu ditampilkan pada LCD.

3.3.3 *Interaural Time Difference*

Setelah melalui penmrosesan pada *cross-correlation*, indeks korelasi tertinggi diproses lebih lanjut menggunakan ITD. Pada metode ini, terdapat beberapa variabel yang perlu dipertimbangkan, salah satunya adalah *time-sampling*. *Time-sampling* terkecil dapat dicari menggunakan variabel jarak sumber suara dan jarak antar mikrofon terkecil. Pada kasus ini, digunakan 20 cm sebagai jarak antar mikrofon dan 30 cm sebagai jarak sumber suara. Sehingga dengan pergeseran 30° dapat ditemukan *delay* menggunakan perhitungan sebagai berikut.

$$delay = \frac{36}{384 \times 10^2} - \frac{26}{384 \times 10^2}$$

$$delay = 9{,}3 \times 10^{-4} - 6{,}7 \times 10^{-4}$$

$$delay = 2{,}6 \times 10^{-4} = 26\,\mu s$$

Sehingga, time sampling yang dibutuhkan sistem adalah kurang dari 26 µs. Mengacu pada persamaan 2.5 dan dengan mengganti beberapa variabel, maka persamaan baru yang ditemukan adalah

$$\theta = \sin^{-1} \frac{(26 \times 10^{-6} \times \sigma) \times 384}{20 \times 10^{-2}}$$

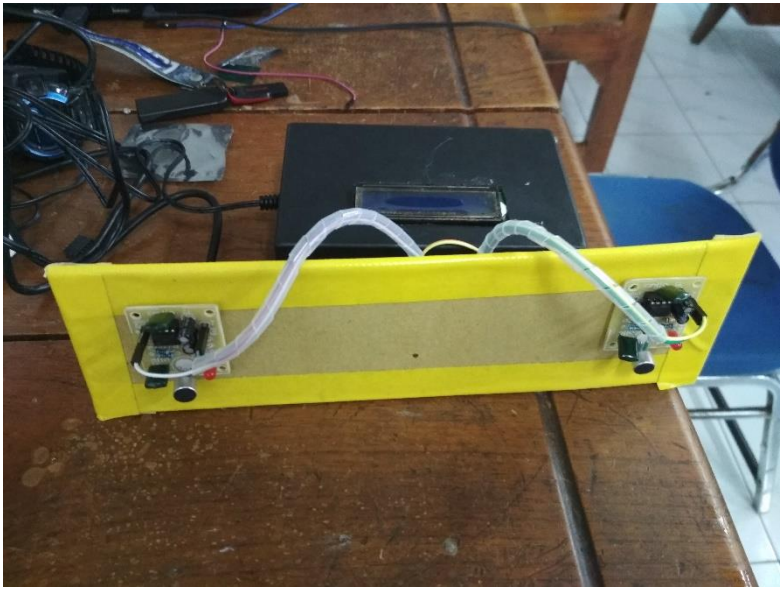
Halaman ini sengaja dikosongkan

BAB IV
PENGUJIAN DAN ANALISIS

Bab ini menjelaskan mengenai hasil pengujian beserta analisa dari data yang didapatkan dari pengujian sistem yang telah dibuat sesuai rancangan di bab sebelumnya. Secara garis besar, pengujian sistem akan dibagi menjadi dua bagian, yaitu pada *hardware*, dan *software*. Pengujian *hardware* berupa pengujian mikrofon dan ADC. Sedangkan pengujian *software* berupa pengujian konversi stokastik, operasi bilangan sederhana, *cross-corellation* dan perbandingan luas wilayah antara metode komputasi stokastik dan komputasi biner.

4.1 Pengujian *Hardware*

Pada pengujian *hardware*, bagian yang diuji adalah mikrofon dan *pre-amplifier* serta ADC DE10-Nano. Pengujian mikrofon akan melibatkan 4 bentuk sinyal yang berbeda yaitu sinyal dengan bentuk sinusoidal, sinyal kotak, sinyal segitiga dan sinyal *sawtooth*.

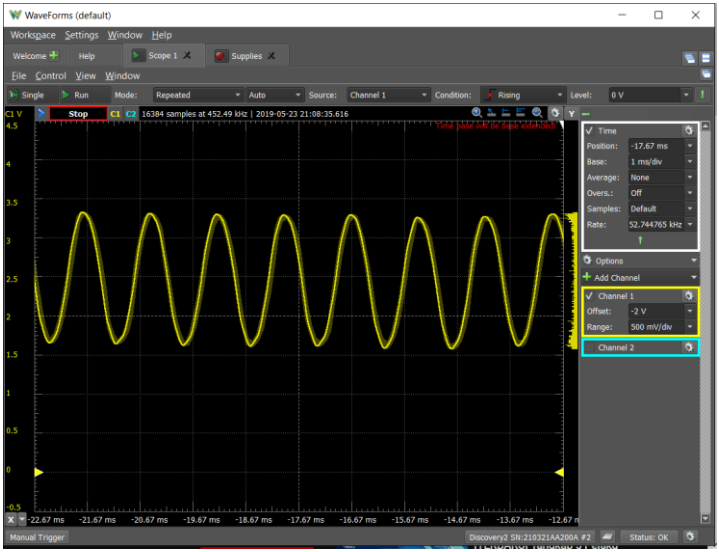


Gambar 4.1 Rangkaian Sistem Lokalisasi Suara

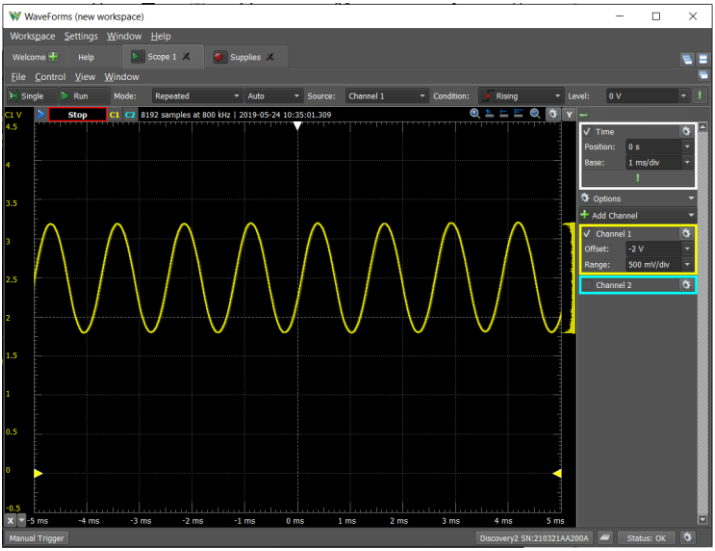
Selanjutnya pengujian ADC yaitu pembacaan sinyal suara sinusoidal yang dijadikan grafik data untuk kedua channel. Sistem pengujian ADC akan memanfaatkan FPGA dan Arduino sebagai prosesor dan pengirim data, sehingga grafik dapat dilihat secara *real-time* melalui terminal serial pada Arduino IDE. Gambar 4.1 menunjukkan bentuk sistem yang dibuat.

4.1.1 Pengujian Mikrofon dan *Pre-amplifier*

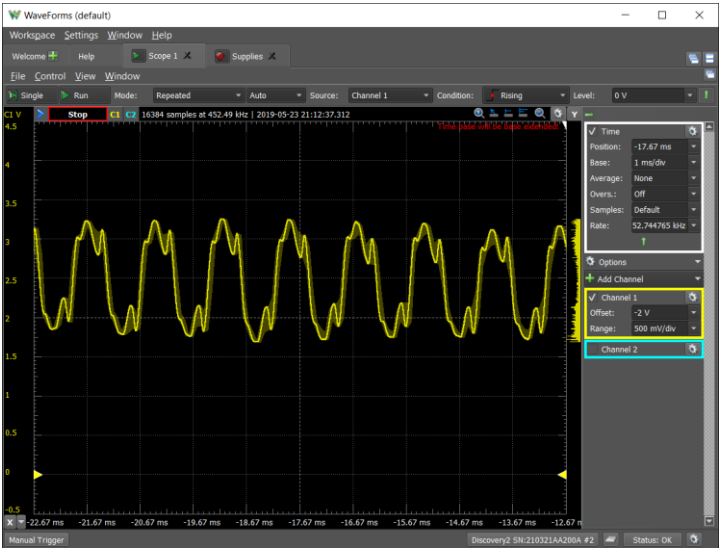
Dalam sistem ini, *hardware* yang digunakan adalah mikrofon kondenser sebagai sensor suara utama, FPGA dan Arduino sebagai prosesor dan data hasil akan ditampilkan di terminal serial. Mikrofon yang digunakan adalah mikrofon kondenser dengan preamplifier berbasis LM358 sehingga output dapat dibaca dan diproses oleh sistem prosesor, Gambar 4.x menunjukkan hasil percobaan dari mikrofon 1 dan 2 dalam memproses sinyal sinusoidal. Suara sinyal sinusoidal dihasilkan menggunakan *frequency generator* dari aplikasi *smartphone*.



Gambar 4.2 Sinyal Output Mikrofon 1 untuk Sinyal Sinusoidal.



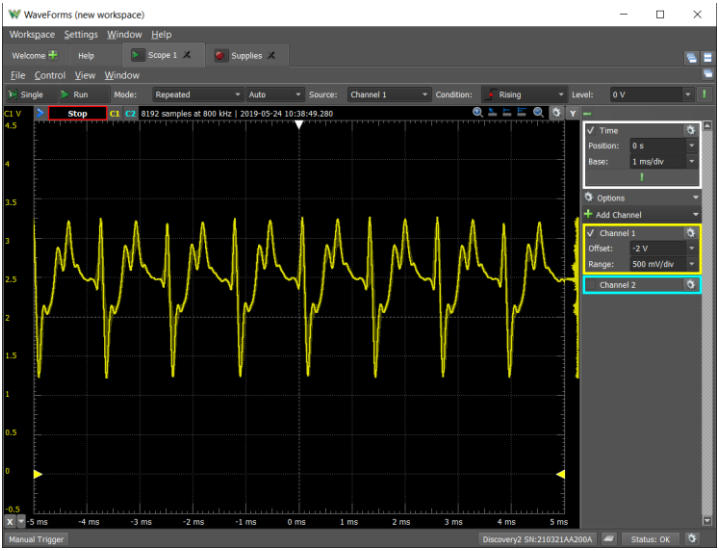
Gambar 4.3 Sinyal Output Mikrofon 2 untuk Sinyal Sinusoidal.



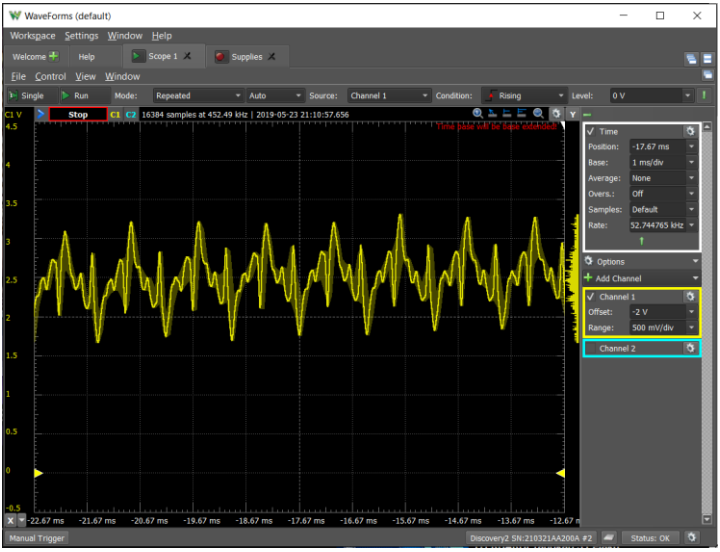
Gambar 4.4 Sinyal Output Mikrofon 1 untuk Sinyal Segitiga.

Gambar 4.5 Sinyal Output Mikrofon 2 untuk Sinyal Segitiga.

Gambar 4.6 Sinyal Output Mikrofon 1 untuk Sinyal *Sawtooth*.



Gambar 4.7 Sinyal Output Mikrofon 2 untuk Sinyal *Sawtooth*.



Gambar 4.8 Sinyal Output Mikrofon 1 untuk Sinyal Kotak.

Gambar 4.9 Sinyal Output Mikrofon 2 untuk Sinyal Kotak.

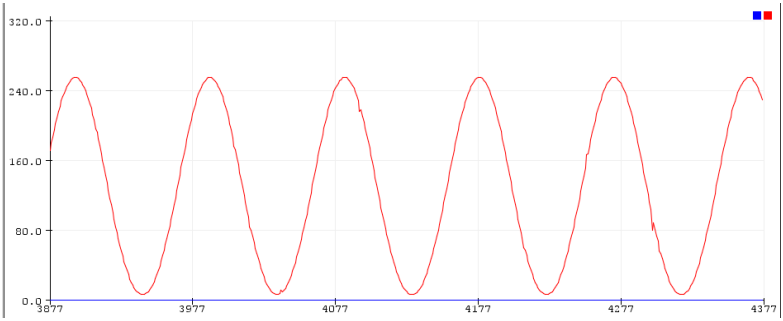
Pada percobaan menggunakan sinyal sinusoidal, hasil yang didapatkan oleh kedua sub-sistem cukup bagus seperti pada gambar 4.2 dan gambar 4.3. Amplitudo yang diraih cukup tinggi dan diharapkan dapat diproses oleh FPGA.

Pada percobaan berikutnya dilakukan menggunakan sinyal suara berbentuk sinyal segitiga (gambar 4.4 dan gambar 4.5), *sawtooth* (gambar 4.6 dan gambar 4.7.) dan kotak (gambar 4.8 dan gambar 4.9) untuk melihat kemampuan sub-sistem dalam menangkap sinyal suara yang memiliki bentuk bentuk tidak natural. Hasil yang didapatkan untuk sinyal segitiga adalah sinyal serupa dengan sinusoidal dengan puncak dan lembah, namun terdapat sedikit *noise* pada kedua puncak yang disebabkan oleh beberapa faktor yaitu kualitas *speaker* dari *smartphone* serta karena ketidakakuratan output suara yang dihasilkan.

Lain halnya dengan sinyal kotak dan *sawtooth*, hasil percobaan menunjukkan hasil yang lebih kacau dibandingkan sinyal segitiga. Sinyal yang dibaca merupakan sinyal tidak beraturan yang periodik. Hal ini disebabkan juga oleh faktor-faktor yang sama yaitu kualitas *speaker* dari *smartphone* dan bentuk sinyal yang cukup jauh dari bentuk natural sinusoidalnya.

4.1.2 Pengujian Analog-to-Digital Converter FPGA

Pengujian ADC dilakukan dengan memanfaatkan *function generator*. Data analog dari *input* akan dibaca oleh ADC lalu menghasilkan nilai digital yang dapat diakses oleh FPGA. Selanjutnya, FPGA akan mengirim data ke Arduino untuk selanjutnya dikirim ke PC melalui komunikasi serial dan ditampilkan sebagai grafik data seperti pada gambar 4.10. Data yang diambil untuk pengujian ADC merupakan data *streaming*, sehingga tidak dialokasikan memori untuk penyimpanan tetap.



Gambar 4.10 Data ADC dengan *Input Function Generator*

4.2 Pengujian *Software*

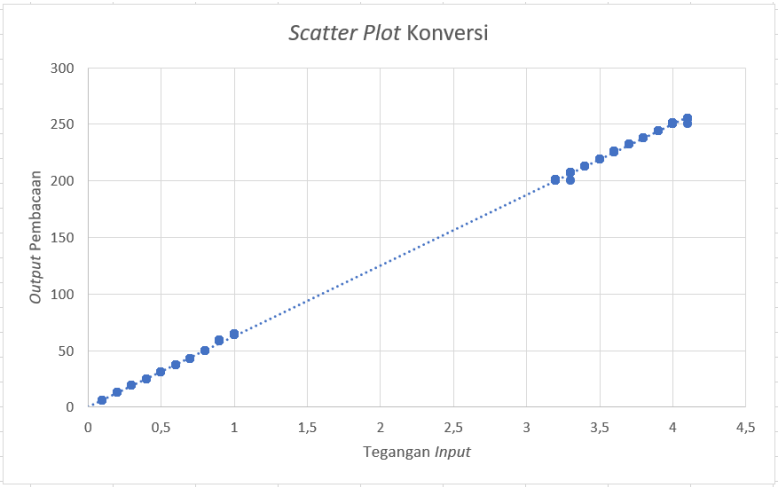
Pada tahap ini akan dilakukan pengujian *software* yang menyusun sistem ini. Pengujian akan dilakukan dalam beberapa tahap, yaitu pengujian tahap konversi bilangan stokastik, tahap perhitungan sederhana, tahap *cross-correlation* dan tahap sintesis utama. Sistem ini memanfaatkan FPGA pemroses data dan Arduino sebagai perantara FPGA dan PC

4.2.1 Pengujian Konversi Bilangan Stokastik

Pada tahap ini akan diuji fungsi konversi dari bilangan biner ke bilangan stokastik yang sudah diimplementasikan pada sistem. Pengujian ini digunakan untuk memastikan pembacaan dari ADC dan hasil konversinya ke bilangan stokastik memiliki hasil yang sesuai tanpa memiliki *error* yang fatal. Prosedur pengujian akan menggunakan dua jenis gelombang yaitu tegangan DC yang konstan dan gelombang sinusoid.

Sinyal analog yang dibaca oleh ADC akan dikonversi menjadi bilangan bilangan stokastik setelah melalui konversi dari resolusi ADC sejumlah 12 bit menjadi 8 bit selanjutnya akan dikonversi menjadi bilangan stokastik lalu akan dikembalikan menjadi bilangan biner.

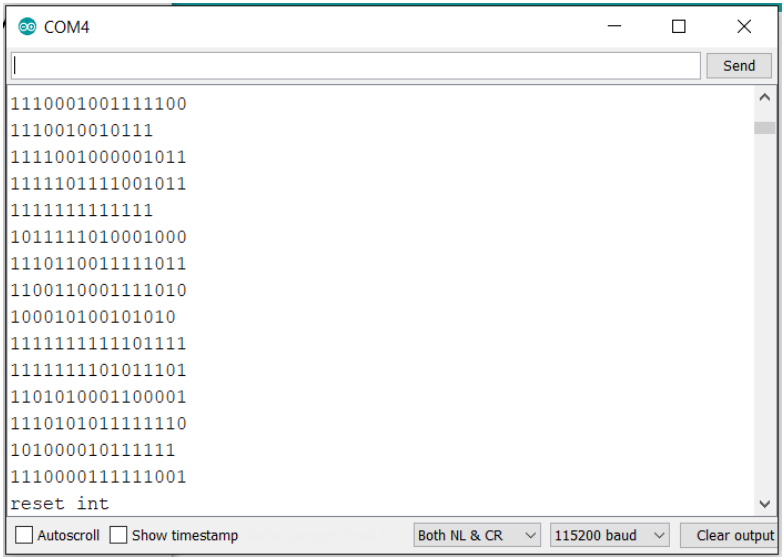
Pengambilan data untuk *input* tegangan DC konstan dilakukan dalam 2 kali *batch* pengambilan. Pengambilan *batch* 1 dilakukan untuk tegangan 100mV sampai 1 V. Sedangkan untuk pengambilan *batch* 2 dilakukan untuk tegangan 3.2 V sampai 4.1 V. Pengambilan data untuk tegangan DC konstan dilakukan untuk 100 sampling untuk tiap nilainya. Dari 100 nilai tersebut akan diselisih antara masing-masing data ADC dan nilai stokastik dan di total untuk 100 sampling data di setiap nilai.



Gambar 4.11 *Scatter Plot* Konversi Bilangan Biner ke Stokastik

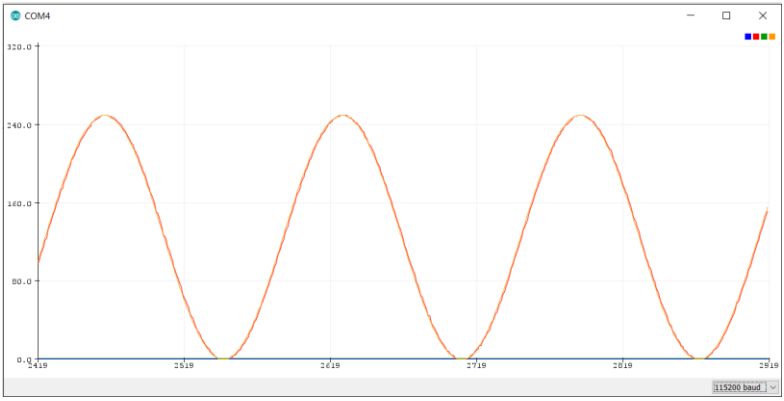
Dari gambar 4.11 dapat disimpulkan bahwa pada nilai-nilai tertentu, konversi stokastik dapat menghasilkan error. Error yang didapatkan dapat diakibatkan karena konversi yang tidak sesuai maupun karena nilai ADC yang fluktuatif. Namun, hal ini merupakan hal yang lumrah terjadi pada komputasi stokastik. Ketidaksesuaian konversi disebabkan oleh proses acak yang merupakan prinsip utama dari sistem komputasi stokastik.

Gambar 4.12 menunjukkan contoh bilangan stokastik yang telah dikonversi dari bilangan biner untuk *input* tegangan 2.5 V. Gambar 4.12 menggambarkan 256 bit bilangan stokastik yang dibagi menjadi 16 baris, 16 bit setiap barisnya. Namun dikarenakan keterbatasan program, beberapa baris hanya memiliki 12 atau 14 bit. Hal ini disebabkan karena beberapa digit didepan bit pertama merupakan angka 0 dan tidak dituliskan oleh program.



Gambar 4.12 Bilangan Stokastik 256 Bit untuk *Input* Tegangan 2.5 V

Pengujian tahap ke dua dilakukan dengan menggunakan sinyal sinusoidal. Output yang dihasilkan berupa grafik yang langsung ditampilkan melalui *serial plotter* Arduino IDE.



Gambar 4.13 Hasil Data untuk Gelombang Sinusoid

Pada pengujian ke dua, hasil output untuk nilai ADC dan bilangan stokastik memiliki data yang nyaris tumpang tindih. Gambar 4.13 membuktikan bahwa bilangan stokastik memiliki akurasi yang cukup bagus dan dapat digunakan tanpa error yang berarti.

4.2.2 Pengujian Operasi Komputasi Stokastik Sederhana

Pengujian pada tahap ini berfungsi sebagai landasan dari proses utama yaitu *cross-correlation*. Sinyal *input* yang digunakan untuk pengujian ini sama dengan tahap konversi yaitu sinyal DC konstan dan gelombang sinusoidal. Operasi yang diuji merupakan perkalian dan penjumlahan, untuk masing-masing *input*.

Pengujian operasi perkalian untuk *input* DC akan dilakukan 20 kali sampling untuk setiap nilai dan akan dibandingkan dengan hasil yang seharusnya.

Tabel 4.1 Perkalian Bilangan Stokastik untuk *Input* 1 Sebesar 2,5V dan *Input* 2 Sebesar 2V

Input 1	*Input* 2	Hasil perkalian stokastik	Hasil perkalian seharusnya
155	124	78	75,07813
156	125	78	76,17188
156	126	78	76,78125
156	124	77	75,5625
156	124	78	75,5625

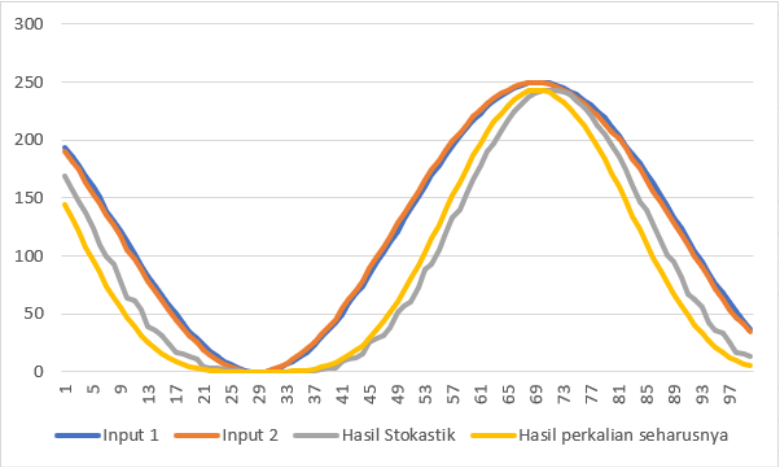
156	125	79	76,17188
156	125	77	76,17188
156	125	77	76,17188
157	125	78	76,66016
157	125	78	76,66016
156	125	78	76,17188
156	126	79	76,78125
155	126	79	76,28906
155	125	78	75,68359
156	125	78	76,17188
156	124	78	75,5625
156	125	78	76,17188
156	125	78	76,17188
157	124	77	76,04688
157	125	78	76,66016

Tabel 4.2 Perkalian Bilangan Stokastik untuk *Input* 1 Sebesar 3,3 V Dan *Input* 2 Sebesar 3,8V

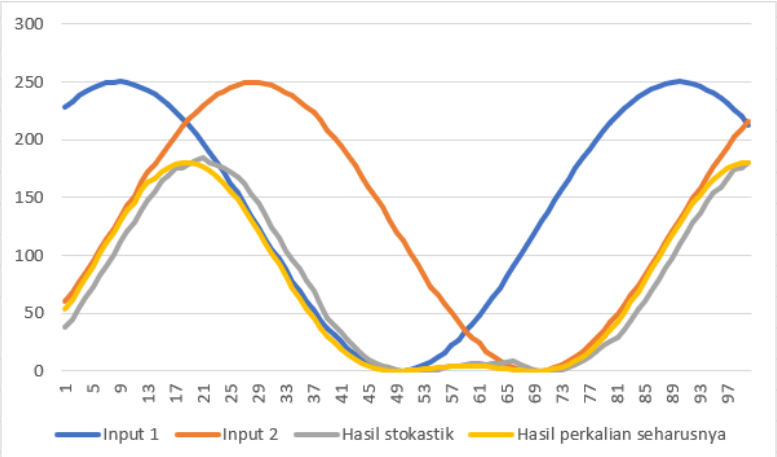
Input 1	*Input* 2	Hasil perkalian stokastik	Hasil perkalian seharusnya
207	237	193	191,6367
206	237	193	190,7109
205	237	193	189,7852
206	237	194	190,7109
206	237	193	190,7109
207	237	192	191,6367
207	237	193	191,6367
206	237	193	190,7109
205	238	194	190,5859
205	237	194	189,7852
206	236	193	189,9063
206	237	193	190,7109
206	237	192	190,7109
206	237	192	190,7109
206	237	193	190,7109
206	237	193	190,7109
206	237	193	190,7109
206	237	193	190,7109
206	237	193	190,7109
206	237	193	190,7109

Pada perkalian untuk *input* sinyal DC sesuai tabel 4.2 dan 4.3, dapat dilihat bahwa nilai perkalian stokastik sedikit melenceng dari nilai perkalian yang seharusnya. Hal ini merupakan error yang lumrah terjadi pada komputasi stokastik [challenge] dikarenakan sifat utama komputasi stokastik yang lebih berupa perkiraan.

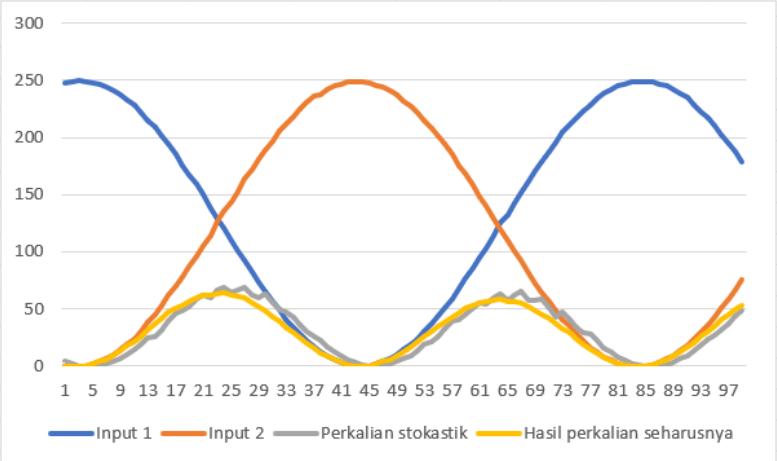
Untuk perkalian dengan *input* gelombang sinusoidal, akan diambil 100 nilai dan dari nilai-nilai tersebut akan dibentuk grafik yang dapat merepresentasikan pengujian tersebut. Dalam grafik akan ditampilkan beberapa parameter, yaitu kedua *input* dan hasil operasi dari kedua gelombang.



Gambar 4.14 Hasil Perkalian Gelombang Sinusoidal dengan Beda Fasa Antar *Input* Sebesar 0 Derajat



Gambar 4.15 Hasil Perkalian Gelombang Sinusoidal dengan Beda Fasa Antar *Input* Sebesar 90 Derajat



Gambar 4.16 Hasil Perkalian Gelombang Sinusoidal dengan Beda Fasa Antar *Input* Sebesar 180 Derajat

Dari hasil yang didapatkan (gambar 4.13, 4.14 dan 4.15) untuk proses perkalian pada gelombang sinusoidal terdapat *delay* dalam perkalian

stokastik sehingga dapat disimpulkan bahwa terdapat *delay* juga perkalian yang menggunakan *input* tegangan DC.
Pengujian operasi penjumlahan untuk *input* DC akan dilakukan 20 kali sampling untuk setiap nilai dan akan dibandingkan dengan hasil yang seharusnya.

Tabel 4.3 Penjumlahan Bilangan Stokastik untuk *Input* 1 Sebesar 2 V Dan *Input* 2 Sebesar 2,5V

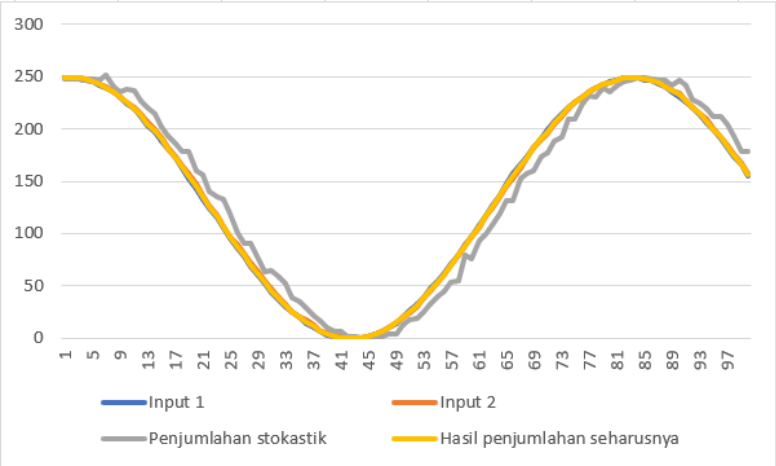
Input 1	*Input* 2	Hasil penjumlahan stokastik	Hasil penjumlahan seharusnya
126	156	134	141
126	156	144	141
125	156	134	140,5
125	157	149	141
124	157	136	140,5
125	156	136	140,5
125	156	136	140,5
124	155	138	139,5
125	155	140	140
125	156	145	140,5
126	156	139	141
126	156	140	141
125	156	145	140,5
124	157	135	140,5
124	157	140	140,5
126	155	136	140,5
126	155	146	140,5
125	156	132	140,5
125	157	147	141
125	156	143	140,5

Tabel 4.4 Perkalian Bilangan Stokastik untuk *Input* 1 Sebesar 3.8 V Dan *Input* 2 Sebesar 3.3V

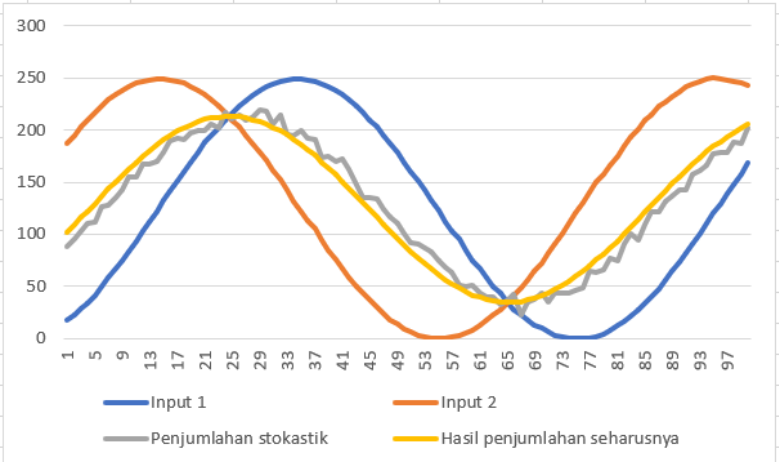
Input 1	*Input* 2	Hasil penjumlahan stokastik	Hasil penjumlahan seharusnya
206	237	214	221,5
206	237	219	221,5
206	237	220	221,5
206	237	223	221,5

206	237	229	221,5
206	237	217	221,5
206	237	220	221,5
206	237	224	221,5
207	237	221	222
207	237	226	222
206	237	220	221,5
205	238	228	221,5
205	237	217	221
206	236	224	221
206	237	214	221,5
206	237	222	221,5
206	237	221	221,5
206	237	220	221,5
207	237	219	222
206	237	227	221,5

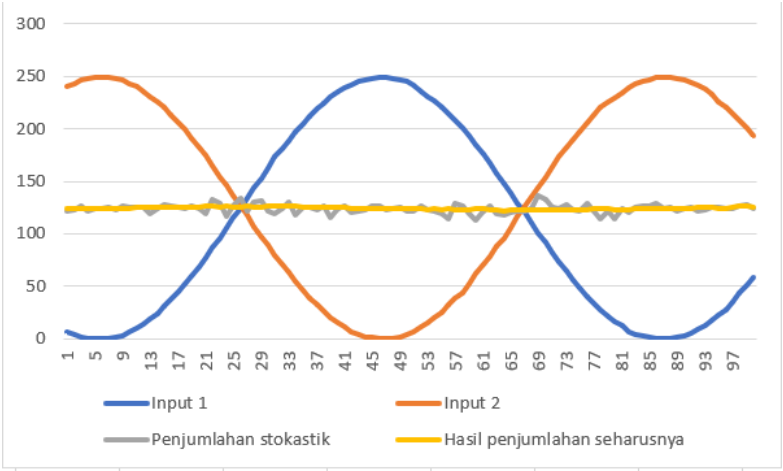
Sama halnya dengan perkalian, pada operasi penjumlahan yang digambarkan pada tabel 4.3 dan 4.4, terdapat error pada operasi penjumlahan stokastik.



Gambar 4.17 Hasil Penjumlahan Gelombang Sinusoidal dengan Beda Fasa Antar *Input* Sebesar 0 Derajat



Gambar 4.18 Hasil Penjumlahan Gelombang Sinusoidal dengan Beda Fasa Antar *Input* Sebesar 90 Derajat



Gambar 4.19 Hasil Penjumlahan Gelombang Sinusoidal dengan Beda Fasa Antar *Input* Sebesar 180 Derajat

Sama halnya dengan perkalian, terdapat *delay* antara *input* dan output pada pengujian gelombang sinusoid (gambar 4.17,4.18 dan 4.19). Sehingga dapat disimpulkan bahwa hal ini juga terjadi pada penjumlahan *input* DC yang mungkin berkontribusi pada terjadinya *error* pada hasil perhitungan.

Dari pengujian pada operasi komputasi stokastik sederhana, selain terdapat *delay* juga terdapat *error-error* kecil di setiap operasi perhitungan stokastik. *Error-error* yang muncul disebabkan oleh prinsip dasar dari komputasi stokastik yang bergantung pada probabilitas.

4.2.3 Pengujian Cross-Correlation

Pada tahap ini akan dilakukan pengujian kemampuan komputasi stokastik untuk melakukan operasi *cross-correlation*. Untuk prosedur yang dilakukan adalah mencari range kemampuan sistem untuk melakukan proses *cross-correlation* untuk mencari indeks korelasi tertinggi dengan *input* gelombang sinusoidal dari *function generator* dan menggunakan mikrofon untuk mendeteksi arah suara.

Pada pengujian dengan *function generator*, variabel yang digunakan adalah frekuensi yang diganti setiap pengujian serta pergeseran fasa dari -180 hingga 180 derajat dengan interval 10 derajat. Data yang dicari dari pengujian ini adalah indeks maksimum dari setiap pergeseran fasa.

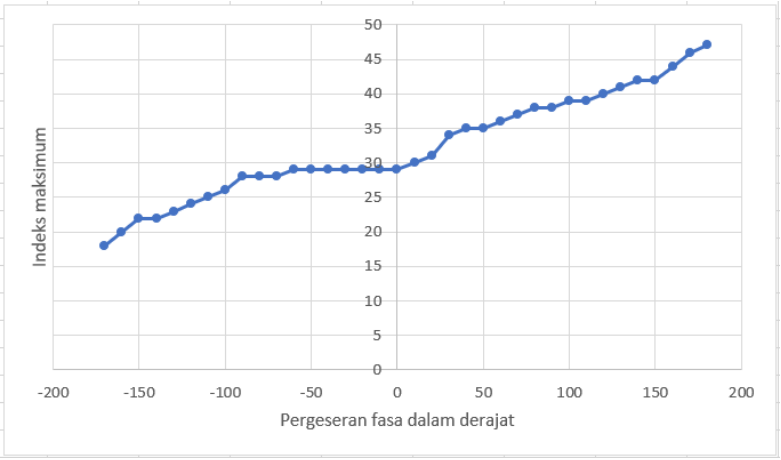
Indeks maksimum yang dimaksud adalah pada indeks ke-berapa korelasi maksimum pada kedua gelombang yang telah digeser fasanya terjadi. Setelah dilakukan *cross-correlation* pada setiap titik, akan dicari titik korelasi tertinggi dan indeks di mana korelasi tertinggi itu terjadi. Dengan memanfaatkan indeks korelasi tertinggi, dapat ditemukan *delay* sinyal dari kedua gelombang *input* yang digunakan.

Delay yang dicari, dapat dihitung memanfaatkan pembagian clock dan timing pengambilan data ADC yang sudah didesain sedemikian rupa pada sistem. Perhitungan detail untuk waktu delay dari setiap index adalah

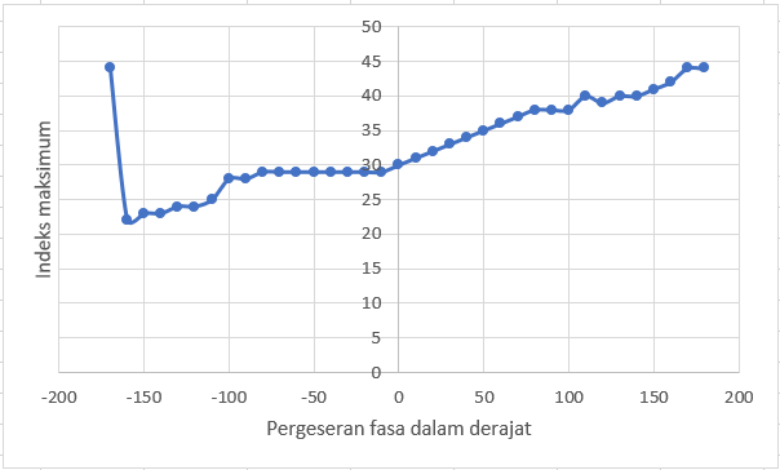
$$\frac{1}{delay} = \frac{System\ Clock}{Divider}$$

$$\frac{1}{delay} = \frac{50\ MHz}{5\ \times 256} = 39{,}062\ KHz$$

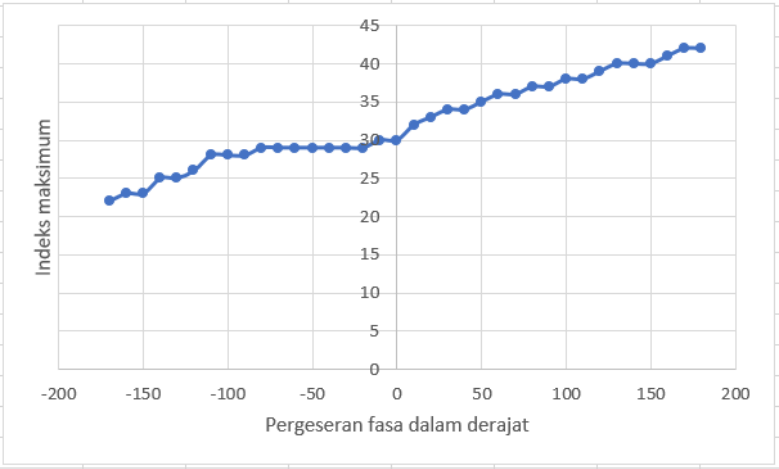
$$delay = \frac{1}{39.062} = 25{,}6\ \mu s$$



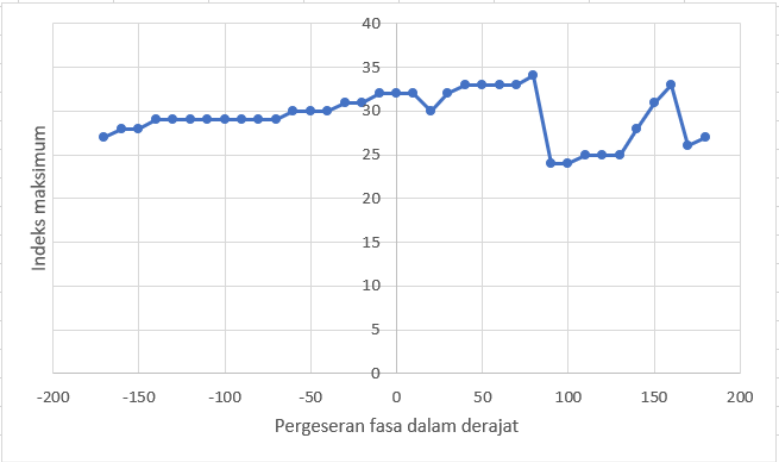
Gambar 4.20 Pencarian Indeks Maksimum untuk Frekuensi 600 Hz



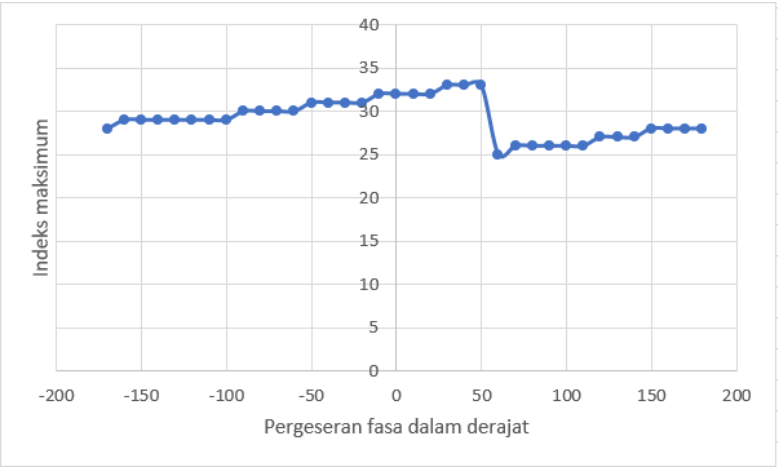
Gambar 4.21 Pencarian Indeks Maksimum untuk Frekuensi 700 Hz



Gambar 4.22 Pencarian indeks maksimum untuk Frekuensi 800 Hz

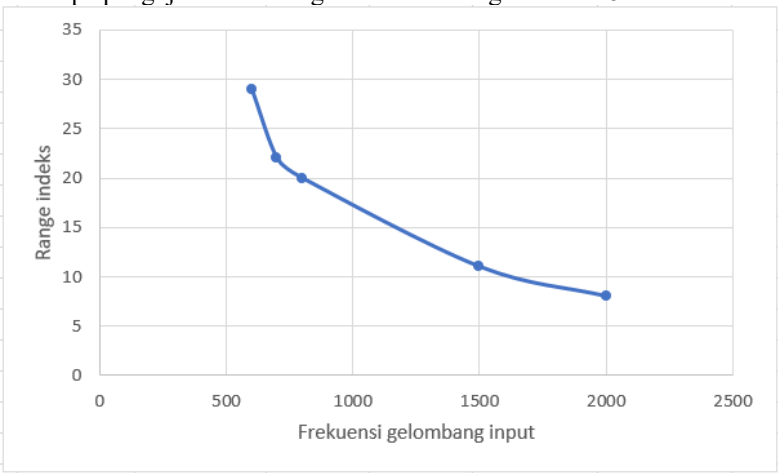


Gambar 4.23 Pencarian Indeks Maksimum untuk Frekuensi 1500 Hz



Gambar 4.24 Pencarian Indeks Maksimum untuk Frekuensi 2000 Hz

Hasil yang didapatkan dari pengujian *cross-correlation* menggunakan *function generator* (gambar 4.21, 4.22, 4.23 dan 4.24) menunjukkan bahwa semakin besar frekuensi sinyal *input*, semakin kecil range pengukuran yang dihasilkan oleh sistem. Hasil pengukuran range dari beberapa pengujian di atas digambarkan oleh gambar 4.25



Gambar 4.25 Range Indeks dari Pengukuran-Pengukuran Indeks Maksimum untuk Setiap Frekuensi

4.2.4 Pengujian Sistem Keseluruhan

Setelah dilakukan pengujian menggunakan *function generator*, pengujian berikutnya digunakan mikrofon sebagai *input*. Variabel yang digunakan adalah arah/sudut dan frekuensi. Pengujian akan dilakukan dengan jarak antar mikrofon sebesar 20 cm dan jarak suara sebesar 30 cm.

Tabel 4.5 Pengujian Sistem dengan Frekuensi 700 Hz

Sudut Pengukuran	Sudut pembacaan	Error Pembacaan
-60	-24	36
-30	-20	10
0	0	0
30	19	11
60	24	36

Tabel 4.6 Pengujian Sistem dengan Frekuensi 800 Hz

Sudut Pengukuran	Sudut pembacaan	Error Pembacaan
-60	-25	35
-30	-21	9
0	0	0
30	20	10
60	26	34

Tabel 4.7 Pengujian Sistem dengan Frekuensi 900 Hz

Sudut Pengukuran	Sudut pembacaan	Error Pembacaan
-60	-21	39
-30	0	30
0	18	18
30	18	12
60	20	40

Tabel 4.8 Pengujian Sistem dengan Frekuensi 1000 Hz

Sudut Pengukuran	Sudut pembacaan	Error Pembacaan
-60	-24	36
-30	-20	10
0	19	19
30	22	8
60	23	37

Tabel 4.9 Pengujian Sistem dengan Frekuensi 1100 Hz

Sudut Pengukuran	Sudut pembacaan	Error Pembacaan
-60	-24	36
-30	-25	5
0	18	18
30	25	5
60	26	34

Tabel 4.10 Pengujian Sistem dengan Frekuensi 1200 Hz

Sudut Pengukuran	Sudut pembacaan	Error Pembacaan
-60	-29	31
-30	-32	2
0	0	0
30	30	0
60	35	25

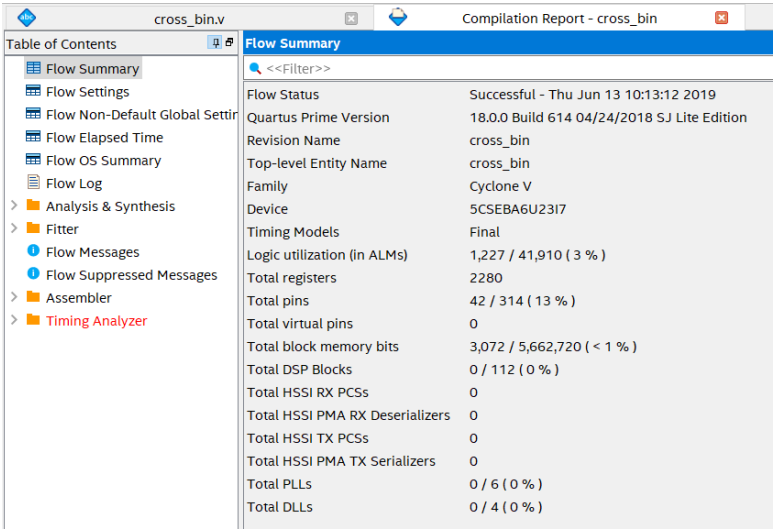
Tabel 4.11 Pengujian Sistem dengan Frekuensi 1300 Hz

Sudut Pengukuran	Sudut pembacaan	Error Pembacaan
-60	-23	37
-30	-18	12
0	20	20
30	22	8
60	27	33

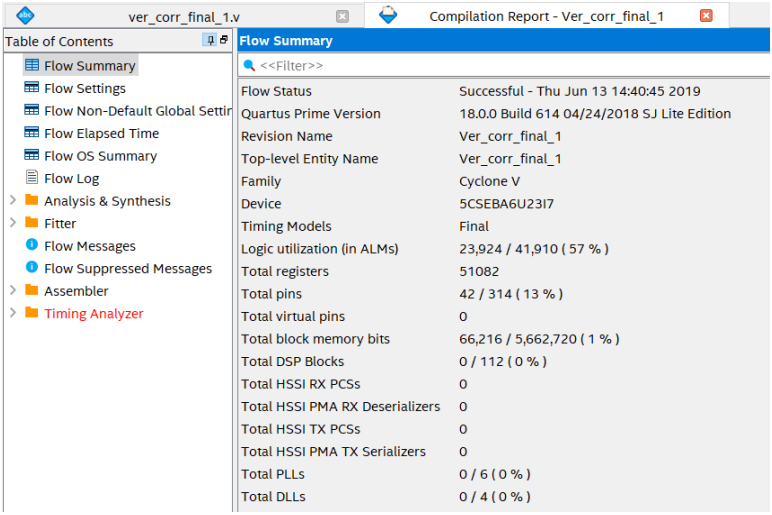
Pada pengujian sistem menggunakan frekuensi suara 700 hingga 1300 Hz (tabel 4.5 sampai tabel 4.11) pembacaan sudut menghasilkan error yang cukup besar, namun dapat memberikan lokalisasi suara secara umum. Mengacu pada data pengujian yang didapatkan, dapat disimpulkan bahwa *cross-correlation* yang diaplikasikan menggunakan komputasi stokastik dapat digunakan untuk mendeteksi arah suara.

4.2.5 Perbandingan *Area Logic Unit*

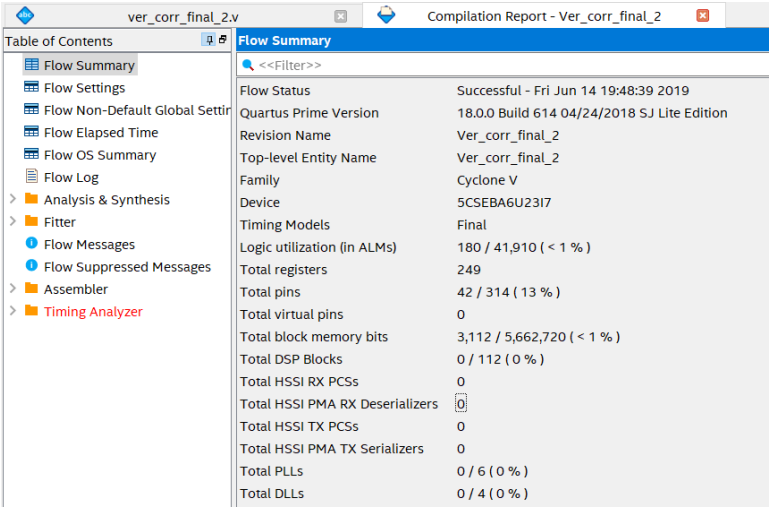
Pada tahap ini akan dibandingkan beberapa hasil sintesis rangkaian untuk menunjukkan seberapa banyak *logic unit* yang digunakan. Dengan mengetahui seberapa banyak *logic unit* yang digunakan, dapat diperkirakan seberapa besar area yang digunakan untuk menjalankan sistem tersebut. Pada tahap ini, akan diuji tiga macam sistem yang memanfaatkan komputasi biner dan stokastik. Ketiga sistem tersebut menjalankan fungsi yang persis sama.



Gambar 4.26 *Compilation Report* untuk Sistem Berbasis Komputasi Biner



Gambar 4.27 *Compilation Report* untuk Sistem Berbasis Komputasi Stokastik dengan Alokasi *Memory* yang Tidak Tepat



Gambar 4.28 *Compilation Report* untuk Sistem Berbasis Komputasi Stokastik dengan Alokasi *Memory* yang Tepat

Mengacu pada *compilation report* di gambar 4.28, 4.29 dan 4.30 dapat disimpulkan bahwa terdapat perbedaan yang cukup besar dalam penggunaan *logic unit* antara sistem komputasi biner dan stokastik. Namun apabila *memory* tidak dialokasikan dengan efisien, komputasi stokastik justru akan jauh lebih merugikan dalam aspek penggunaan *logic unit* daripada komputasi biner.

4.3 Analisis

Analisa yang dapat diperoleh dari hasil-hasil pengujian dengan perancangan pada bab sebelumnya antara lain:

1. Dalam penelitian ini, digunakan sistem komputasi yang non-konvensional yaitu komputasi stokastik. Sebuah komputasi yang menggunakan probabilitas sebagai faktor utama dalam komputasi.

2. Besar sinyal yang masuk pada sistem sangat berpengaruh pada berhasil atau tidaknya lokalisasi suara. Jarak serta

intensitas suara berpengaruh cukup besar pada sistem komputasi stokastik yang pada dasarnya memiliki *error* bawaan, sehingga sangat diperlukan hasil yang benar-benar konklusif untuk sistem ini dapat bekerja.

3. Semakin tinggi frekuensi sinyal *input*, semakin kecil *range* pengukuran yang didapat. Hal ini disebabkan semakin tinggi frekuensi, semakin pendek panjang gelombang. Sehingga pada pergeseran fasa maksimum pergeseran gelombang frekuensi tinggi tidak sejauh gelombang frekuensi rendah.

4. Dilakukan percobaan menggunakan 2 tingkat penguatan, apabila penguatan *pre-amplifier* terlalu rendah, operasi *time-delay analysis* pada *cross-correlation* akan menghasilkan error konstan dan sistem tidak berjalan.

5. Frekuensi yang dapat digunakan agar sistem dapat bekerja ada pada kisaran 700 Hz hingga 1400 Hz, dikarenakan pada frekuensi tersebut intensitas suara dirasa cukup tinggi untuk diproses.

6. Dalam mendesain sistem FPGA pengaturan *resource* dan *memory* sangat diperlukan, dikarenakan perbedaan kecil pada program bisa mempengaruhi *logic utilization* secara drastis.

Halaman ini sengaja dikosongkan

BAB V
PENUTUP

5.1 Kesimpulan

Pada penilitian ini telah dirancang dan dibuat lokalisasi suara berbasis komputasi stokastik menggunakan FPGA. Berdasarkan data hasil uji implementasi komputasi stokastik pada sistem lokalisasi suara, diperoleh beberapa kesimpulan antara lain:

1. Sistem komputasi stokastik tebukti dapat diaplikasikan pada metode *cross-correlation* yang menjadi dasar dari sistem lokalisasi suara.
2. *Cross-correlation* dan Sistem lokalisasi suara merupakan aplikasi yang baru dari sistem komputasi stokastik.
3. Lokalisasi suara pada sistem ini memiliki error yang cukup besar, hal ini disebabkan karena komputasi stokastik lebih bersifat estimasi daripada perhitungan tepat.
4. Metode *cross-correlation* menghabiskan cukup banyak *resource* sebagai penyimpanan vektor *input* dan *output* apabila disimpan dalam bentuk bilangan stokastik.
5. Dalam penggunaan komputasi stokastik, sebisa mungkin hindari penyimpanan bilangan stokastik dalam memori.

5.2 Saran

Sebagai sarana pengembangan sistem Lokalisasi Suara berbasis Komputasi Stokastik dengan FPGA, maka terdapat beberapa saran yang dapat disampaikan penulis, beberapa diantaranya:

1. Rangkaian pre-amp dapat lebih disempurnakan agar tidak membawa bias-bias yang dapat mengacaukan perhitungan.
2. Optimasi program Verilog agar lebih efektif dan efisien, program yang digunakan pada sistem ini masih sangat bisa untuk lebih disempurnakan kembali.

Demikian saran yang dapat penulis sampaikan. Semoga dapat bermanfaat ke depannya

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] A. Alaghi, W. Qian, dan J. P. Hayes, "The Promise and Challenge of Stochastic Computing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, hlm. 1–1, 2017.

[2] M. Asfari, M. Rivai, dan T. Tasripan, "Penentuan Arah Sumber Suara dengan Metode Interaural Time Difference menggunakan Mikrokontroler STM32F4," *J. Tek. ITS*, vol. 6, no. 2, Okt 2017.

[3] J. P. Hayes, "Introduction to stochastic computing and its challenges," 2015, hlm. 1–3.

[4] S. M. Trimberger, *Field-Programmable Gate Array Technology*. Boston, MA: Springer US, 1994.

[5] D. Bingqian dan L. Fangmin, "An active sound localization method based on mobile phone," dalam *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, Changchun, China, 2016, hlm. 846–850.

[6] S. Palnitkar, *Verilog HDL: a guide to digital design and synthesis*. Mountain View, Calif.: SunSoft Press [u.a.], 1996.

[7] "Arduino Nano." [Daring]. Tersedia pada: https://store.arduino.cc/usa/arduino-nano. [Diakses: 27-Mei-2019].

[8] A. Klein, *Stream ciphers*. London: Springer, 2013.

Halaman ini sengaja dikosongkan

LAMPIRAN I
HDL SISTEM BERBASIS KOMPUTASI STOKASTIK

```
module Ver_corr_final_2(


        //////////// ADC //////////
        output                          ADC_CONVST,
        output                          ADC_SCK,
        output                          ADC_SDI,
        input                           ADC_SDO,

        //////////// ARDUINO //////////
        inout           reg   [15:0]            ARDUINO_IO,
        inout                           ARDUINO_RESET_N,
        output          reg
        ARDUINO_INT,
        output          reg       [2:0]   SEL,

        //////////// CLOCK //////////
        input                           FPGA_CLK1_50,
        input                           FPGA_CLK2_50,
        input                           FPGA_CLK3_50,

        //////////// KEY //////////
        input               [1:0]       KEY,

        //////////// LED //////////
        output          reg   [7:0]             LED,

        //////////// SW //////////
        input               [3:0]       SW
);



//=======================================================
=====
//  REG/WIRE declarations
```

```
//=====================================================
=====
parameter c_NUM_BITS=8;
        //clock divider
        reg     [3:0]   cnt;
        reg     [10:0]  cnt1;
        reg     clk;
        reg     clk1;
        //converting use
        reg [7:0] lfsr_buff[2:0];
        reg     compi[1:0];
        //adc use
        reg [11:0] outa;
        reg [11:0] outb;
        //temporary register
        //multiplication
        reg [7:0] conva;
        reg [7:0] convb;
        reg [20:0] countm;
        reg stres;
        //storage array
        reg [7:0] sampa [127:0];
        reg [7:0] sampb [127:0];
        reg [20:0]          countc [100:0];
        //loop tracking
        reg [7:0] repi;
        reg [7:0] repo;
        reg [7:0] repm;
        reg [7:0] repa;
        //index tracking
        reg [7:0] indexi;
        reg [7:0] indexo;
        reg [7:0] indexm;
        //reg [2:0] indexa;
        reg [5:0] indexc;
        //reg [7:0] indexr;

        reg [7:0] bbuff [1:0];
        reg [20:0] maxc;
```

```
        reg [6:0] maxi;
        reg contrig;
        reg protrig;
        reg addtrig;
        reg intrig;
        wire clk_out;
        wire [7:0] lfsr_out;



//=============================================================
=====
//  Structural coding
//=============================================================
=====
ADC_TRY(.CLOCK (FPGA_CLK1_50),
                .RESET  (!KEY[0]),
                .CH0    (outa),
                .CH1    (outb),
                .ADC_SCLK       (ADC_SCK),
                .ADC_CS_N       (ADC_CONVST),
                .ADC_DOUT       (ADC_SDO),
                .ADC_DIN                (ADC_SDI));

always@(posedge FPGA_CLK1_50)
begin
if(cnt<5)
        cnt<=cnt+1;
        else
        begin
        cnt<=0;
        //LED[7]<=~LED[7];
        clk<=~clk;

        end

if(cnt1<600)
        cnt1<=cnt1+1;
```

```
            else
            begin
            cnt1<=0;
            //LED[7]<=~LED[7];
            clk1<=~clk1;

            end


            end
//assign LED[7]= clk;
assign clk_out=clk;
LFSR #(.NUM_BITS(c_NUM_BITS)) LFSR_inst //inst digunakan agar
modul LFSR
        (.i_Clk(clk),                                                   //dapat
digunakan beberapa kali dalam satu modul
        .i_Enable(1'b1),
        .i_Seed_DV(1'b0),
        .i_Seed_Data({c_NUM_BITS{1'b0}}), // Replication
        .o_LFSR_Data(lfsr_out),
        .o_LFSR_Done()
        );


always@(posedge clk)
begin
if(intrig==0)
begin

            repi<=repi+1;
            if(repi==0)
                        begin
                                    sampa[indexi]<=outa[11:4];
                                    sampb[indexi]<=outb[11:4];
                                    //LED<=indexi;
                                    //LED<=stocout;

                        end
            if(repi==20)
```

```
                indexi=indexi+1'b1;
        if(indexi==127&&KEY[1]==1)
                begin
                intrig<=1;
                protrig<=1'b1;//process trigger
                end

end

        if(KEY[1]==0)
                intrig<=0;

        //process
        if(protrig>0&&intrig>0)
        begin
                // LFSR De-correlator
                lfsr_buff[2]<=lfsr_out;
                lfsr_buff[1]<=lfsr_buff[2];
                lfsr_buff[0]<=lfsr_buff[1];

                compi[0] <= (conva>lfsr_out) ? 1:0;

                compi[1] <= (convb>lfsr_buff[0]) ? 1:0;

                stres<=compi[0] & compi[1];

                countm<=countm+stres;
                //LED[7:0]<=repi[7:0];
                repm<=repm+1'b1;
                if(repm==0)
                        begin
                                conva<=sampa[indexm+indexc];
                                convb<=sampb[32+indexc];
                                if(indexc==63)
                                begin
                                        indexm<=indexm+1;
                                        countc[indexm]<=countm;
                                        if(countm>maxc)
                                        begin
```

```
                                                maxc<=countm;
                                                maxi<=indexm;
                                        end
                                        countm<=0;
                                        contrig<=1;
                                end
                                //countm<=0;
                                //LED<=repm;
                                //LED[1]<=1;
                        end
                if(repm==50)
                        begin
                        indexc=indexc+1'b1;
                        if(indexm==64)
                                begin
                                indexm<=0;
                                maxc<=0;
                                maxi<=0;
                                end
                        end
        end


        //Master reset
        /*if(KEY[1]==0)
        begin
                for(*/
end
/*if(countc[indexo]>maxc+10)
                                        begin

        maxc<=countc[indexo];
                                                maxi<=indexo;
                                        end
                                if(indexo==127)
                                        begin
                                                maxc<=0;
                                                maxi<=0;
                                        end*/
```

```
always@(posedge clk1)
begin
//stoc to bin convert
        if(contrig>0)
        begin

                /*compo[2] <= (stocoutm[0]>0) ? 1:0;
                stocoutm <= stocoutm>>1'b1;
                counto[2] <= counto[2]+compo[2];*/
                repo=repo+1'b1;
                if(repo==0)
                        begin
                                if(SW[0]==1)
                                        begin

        ARDUINO_IO[7:0]<=~sampa[indexo];

        bbuff[0]<=~sampb[indexo];

        bbuff[1]<=~countc[indexo];
                                        end
                                else
                                if(SW[1]==1)
                                        begin

        ARDUINO_IO[7:0]<=~sampa[indexo];

        bbuff[0]<=~sampb[indexo];
                                        bbuff[1]<=~maxi;
                                        //bbuff[1]<=~counto[2];
                                        end
                                else
                                if(SW[2]==1)
                                        begin

        ARDUINO_IO[7:0]<=~sampa[indexo];

        bbuff[0]<=~sampb[indexo];
```

```
                                        bbuff[1]<=~maxc;
                                        end
                                else
                                if(SW[3]==1)
                                        begin

ARDUINO_IO[7:0]<=~maxi;
                                        bbuff[0]<=~maxc;

bbuff[1]<=~countc[indexo];
                                        end
                                //LED[2]<=1;
                                ARDUINO_INT<=1;
                                end
        if(repo==40)
                begin
                                ARDUINO_INT<=0;
                                SEL<=3'b101;
                                indexo<=indexo+1'b1;
                end
        if(repo==80)
                begin
                                ARDUINO_INT<=1;
                                ARDUINO_IO[7:0]<=bbuff[0];
                end
        if(repo==120)
                begin
                                ARDUINO_INT<=0;
                                SEL<=3'b110;
                end
        if(repo==160)
                begin
                                ARDUINO_INT<=1;
                                ARDUINO_IO[7:0]<=bbuff[1];
                end
        if(repo==200)
                begin
                                ARDUINO_INT<=0;
                                SEL<=3'b011;
```

```
                end

            end


end
endmodule
```

Halaman ini sengaja dikosongkan

LAMPIRAN II
HDL SISTEM BERBASIS KOMPUTASI STOKASTIK (ALOKASI MEMORI SALAH)

```
module Ver_corr_final_1(

	//////////// ADC //////////
	output		          		ADC_CONVST,
	output		          		ADC_SCK,
	output		          		ADC_SDI,
	input		          		ADC_SDO,

	//////////// ARDUINO //////////
	inout		reg	[15:0]		ARDUINO_IO,
	inout		          		ARDUINO_RESET_N,
	output		reg		
ARDUINO_INT,
	output		reg	[2:0]	SEL,

	//////////// CLOCK //////////
	input		          		FPGA_CLK1_50,
	input		          		FPGA_CLK2_50,
	input		          		FPGA_CLK3_50,

	//////////// KEY //////////
	input		[1:0]		KEY,

	//////////// LED //////////
	output		reg	[7:0]		LED,

	//////////// SW //////////
	input		[3:0]		SW
);
```

```
//=================================================
=====
//  REG/WIRE declarations
//=================================================
=====
parameter c_NUM_BITS=8;
        //clock divider
        reg     [3:0]   cnt;
        reg     [10:0]  cnt1;
        reg     clk;
        reg     clk1;
        //converting use
        reg [7:0] lfsr_buff[2:0];
        reg     compi[1:0];
        reg     compo[2:0];
        reg [7:0] counto[2:0];
        reg [255:0] stoca;
        reg [255:0] stocb;
        reg [255:0] stocouta;
        reg [255:0] stocoutb;
        reg [255:0] stocoutm;
        //adc use
        reg [11:0] outa;
        reg [11:0] outb;
        reg [7:0] buff[1:0];
        //temporary register
        //multiplication
        reg [255:0] stmula;
        reg [255:0] stmulb;
        reg [20:0] countm;
        reg stres;
        //storage array
        reg [255:0] stouta [127:0];
        reg [255:0] stoutb [127:0];
        reg [255:0] stoutm [7:0];
```

```
        reg [20:0]          countc [100:0];
        //loop tracking
        reg [7:0] repi;
        reg [7:0] repo;
        reg [7:0] repm;
        reg [7:0] repa;
        //index tracking
        reg [7:0] indexi;
        reg [7:0] indexo;
        reg [7:0] indexm;
        reg [2:0] indexa;
        reg [5:0] indexc;
        reg [7:0] indexr;

        reg [7:0] bbuff [1:0];
        reg [20:0] maxc;
        reg [6:0] maxi;
        reg contrig;
        reg protrig;
        reg addtrig;
        reg intrig;
        wire clk_out;
        wire [7:0] lfsr_out;



//=================================================
======
//  Structural coding
//=================================================
======
ADC_TRY(.CLOCK (FPGA_CLK1_50),
                .RESET  (!KEY[0]),
                .CH0    (outa),
                .CH1    (outb),
```

```
                .ADC_SCLK       (ADC_SCK),
                .ADC_CS_N       (ADC_CONVST),
                .ADC_DOUT       (ADC_SDO),
                .ADC_DIN                (ADC_SDI));

always@(posedge FPGA_CLK1_50)
begin
if(cnt<5)
        cnt<=cnt+1;
        else
        begin
        cnt<=0;
        //LED[7]<=~LED[7];
        clk<=~clk;

        end

if(cnt1<600)
        cnt1<=cnt1+1;
        else
        begin
        cnt1<=0;
        //LED[7]<=~LED[7];
        clk1<=~clk1;

        end


        end
//assign LED[7]= clk;
assign clk_out=clk;
LFSR #(.NUM_BITS(c_NUM_BITS)) LFSR_inst //inst digunakan agar
modul LFSR
      (.i_Clk(clk),                                     //dapat
digunakan beberapa kali dalam satu modul
       .i_Enable(1'b1),
```

```
        .i_Seed_DV(1'b0),
        .i_Seed_Data({c_NUM_BITS{1'b0}}), // Replication
        .o_LFSR_Data(lfsr_out),
        .o_LFSR_Done()
        );


always@(posedge clk)
begin
if(intrig==0)
begin
            // LFSR De-correlator
            lfsr_buff[2]<=lfsr_out;
            lfsr_buff[1]<=lfsr_buff[2];
            lfsr_buff[0]<=lfsr_buff[1];
            // bin to stoc convert[ch0]
            compi[0] <= (buff[0]>lfsr_out) ? 1:0;
            stoca<=stoca<<1'b1;
            stoca[0]<=compi[0];

            // bin to stoc convert[ch1]
            compi[1] <= (buff[1]>lfsr_buff[0]) ? 1:0;
            stocb<=stocb<<1'b1;
            stocb[0]<=compi[1];


            repi<=repi+1;
            if(repi==0)
                        begin
                                    stouta[indexi]<=stoca;
                                    stoutb[indexi]<=stocb;
                                    buff[0]<=outa[11:4];
                                    buff[1]<=outb[11:4];
                                    //LED<=indexi;
                                    //LED<=stocout;
```

```
                end
        if(repi==20)
                indexi=indexi+1'b1;
        if(indexi==127&&KEY[1]==1)
                begin
                intrig<=1;
                protrig<=1'b1;//process trigger
                end

end

        if(KEY[1]==0)
                intrig<=0;

        //process
        if(protrig>0&&intrig>0)
        begin
                stmula<=stmula>>1'b1;
                stmulb<=stmulb>>1'b1;
                stres<=stmula[0] & stmulb[0];
                countm<=countm+stres;
                //LED[7:0]<=repi[7:0];
                repm<=repm+1'b1;
                if(repm==0)
                        begin
                                stmula<=stouta[indexm+indexc];
                                stmulb<=stoutb[32+indexc];
                                if(indexc==63)
                                begin
                                        indexm<=indexm+1;
                                        countc[indexm]<=countm;
                                        if(countm>maxc)
                                        begin
                                                maxc<=countm;
                                                maxi<=indexm;
                                        end
```

```
                                        countm<=0;
                                        contrig<=1;
                                end
                                //countm<=0;
                                //LED<=repm;
                                //LED[1]<=1;
                        end
                if(repm==50)
                        begin
                        indexc=indexc+1'b1;
                        if(indexm==64)
                                begin
                                indexm<=0;
                                maxc<=0;
                                maxi<=0;
                                end
                        end
        end


        //Master reset
        /*if(KEY[1]==0)
        begin
                for(*/
end
/*if(countc[indexo]>maxc+10)
                                        begin

        maxc<=countc[indexo];
                                                maxi<=indexo;
                                        end
                                if(indexo==127)
                                        begin
                                                maxc<=0;
                                                maxi<=0;
                                        end*/
```

```
always@(posedge clk1)
begin
//stoc to bin convert
        if(contrig>0)
        begin

                stocouta <= stocouta>>1'b1;
                counto[0] <= counto[0]+stocouta[0];


                stocoutb <= stocoutb>>1'b1;
                counto[1] <= counto[1]+stocoutb[1];

                /*compo[2] <= (stocoutm[0]>0) ? 1:0;
                stocoutm <= stocoutm>>1'b1;
                counto[2] <= counto[2]+compo[2];*/
                repo=repo+1'b1;
                if(repo==0)
                        begin
                                stocouta[255:0]<=stouta[indexo];
                                stocoutb[255:0]<=stoutb[indexo];
                                //stocoutc[255:0]<=stoutm[indexo]
                                if(SW[0]==1)
                                        begin

        ARDUINO_IO[7:0]<=~counto[0];
                                        bbuff[0]<=~counto[1];

        bbuff[1]<=~countc[indexo];
                                        end
                                else
                                if(SW[1]==1)
                                        begin

        ARDUINO_IO[7:0]<=~counto[0];
```

```
                                        bbuff[0]<=~counto[1];
                                        bbuff[1]<=~maxi;
                                        //bbuff[1]<=~counto[2];
                                        end
                                else
                                if(SW[2]==1)
                                        begin

ARDUINO_IO[7:0]<=~counto[0];
                                        bbuff[0]<=~counto[1];
                                        bbuff[1]<=~maxc;
                                        end
                                else
                                if(SW[3]==1)
                                        begin

ARDUINO_IO[7:0]<=~maxi;
                                        bbuff[0]<=~maxc;

bbuff[1]<=~countc[indexo];
                                        end
                                counto[0]<=0;
                                counto[1]<=0;
                                counto[2]<=0;
                                //LED[2]<=1;
                                ARDUINO_INT<=1;
                                end
        if(repo==40)
                begin
                                ARDUINO_INT<=0;
                                SEL<=3'b101;
                                indexo<=indexo+1'b1;
                end
        if(repo==80)
                begin
                                ARDUINO_INT<=1;
```

```
                        ARDUINO_IO[7:0]<=bbuff[0];
                end
            if(repo==120)
                begin
                        ARDUINO_INT<=0;
                        SEL<=3'b110;
                end
            if(repo==160)
                begin
                        ARDUINO_INT<=1;
                        ARDUINO_IO[7:0]<=bbuff[1];
                end
            if(repo==200)
                begin
                        ARDUINO_INT<=0;
                        SEL<=3'b011;
                end

    end


end
endmodule
```

LAMPIRAN III
HDL SISTEM BERBASIS KOMPUTASI BINER

```
module cross_bin(

        //////////// ADC //////////
        output                  ADC_CONVST,
        output                  ADC_SCK,
        output                  ADC_SDI,
        input                   ADC_SDO,

        //////////// ARDUINO //////////
        inout     reg   [15:0]  ARDUINO_IO,
        output    reg
        ARDUINO_INT,
        inout                   ARDUINO_RESET_N,
        output    reg     [2:0] SEL,

        //////////// CLOCK //////////
        input                   FPGA_CLK1_50,
        input                   FPGA_CLK2_50,
        input                   FPGA_CLK3_50,

        //////////// KEY //////////
        input             [1:0] KEY,

        //////////// LED //////////
        output   reg      [7:0] LED,

        //////////// SW //////////
        input             [3:0] SW

);
```

```
//=============================================================
=====
//  REG/WIRE declarations
//=============================================================
=====
reg [11:0] outa;
reg [11:0] outb;
reg [2:0] cnt;
reg [9:0] cnt1;
reg clk;
reg clk1;
reg [7:0] ina [127:0];
reg [7:0] inb [127:0];
reg [20:0] count;
reg [20:0] count_out [127:0];
reg [7:0] samp;
reg [20:0] maxc;
reg [7:0] maxi;
reg [6:0] indexi;
reg [6:0] indexa;
reg [5:0] indexb;
reg [7:0] indexo;
reg [7:0] bbuff [1:0];
reg [7:0] repo;
reg protrig;
reg intrig;
reg contrig;



//=============================================================
=====
//  Structural coding
//=============================================================
=====
ADC_TRY(.CLOCK (FPGA_CLK1_50),
```

```
                .RESET  (!KEY[0]),
                .CH0    (outa),
                .CH1    (outb),
                .ADC_SCLK       (ADC_SCK),
                .ADC_CS_N       (ADC_CONVST),
                .ADC_DOUT       (ADC_SDO),
                .ADC_DIN                (ADC_SDI));

always@(posedge FPGA_CLK1_50)
begin
if(cnt<5)
        cnt<=cnt+1;
        else
        begin
        cnt<=0;
        //LED[7]<=~LED[7];
        clk<=~clk;
        end

if(cnt1<600)
        cnt1<=cnt1+1;
        else
        begin
        cnt1<=0;
        //LED[7]<=~LED[7];
        clk1<=~clk1;
        end
end

always@(posedge clk)
begin
if(intrig==0)
begin
if(samp==0)
        begin
                ina[indexi]<=outa;
```

```
                inb[indexi]<=outb;
        end

        samp<=samp+1;
if(samp==120)
        indexi<=indexi+1;

if(indexi==127&&KEY[1]==1)
        begin
                intrig<=1;
                protrig<=1;
        end
end

if(intrig>0&&KEY[1]==0)
        intrig<=0;

if(protrig>0&&intrig>0)
begin
  count<=count+(ina[32+indexa]*inb[indexa+indexb]);
  indexa<=indexa+1;
  if(indexa==63)
        begin
        indexb=indexb+1;
        if(count>maxc)
                begin
                maxc<=count;
                maxi<=indexb;
                end
        count_out[indexb]<=count;
        LED<=indexb;
        count<=0;
        contrig<=1;
        end

  if(indexb==64)
```

```
        begin
        indexb<=0;
        maxc<=0;
        maxi<=0;
        end

end
end

always@(posedge clk1)
begin
        if(contrig>0)
                begin
                repo=repo+1'b1;
                if(repo==0)
                        begin
                                if(SW[0]==1)
                                        begin

        ARDUINO_IO[7:0]<=~ina[indexo];
                                        bbuff[0]<=~inb[indexo];

        bbuff[1]<=~count_out[indexo];
                                        end
                                else
                                if(SW[1]==1)
                                        begin

        ARDUINO_IO[7:0]<=~ina[indexo];
                                        bbuff[0]<=~inb[indexo];
                                        bbuff[1]<=~maxi;
                                        //bbuff[1]<=~counto[2];
                                        end
                                else
                                if(SW[2]==1)
                                        begin
```

```
ARDUINO_IO[7:0]<=~ina[indexo];
                                        bbuff[0]<=~inb[indexo];
                                        bbuff[1]<=~maxc;
                                        end
                                else
                                if(SW[3]==1)
                                        begin

ARDUINO_IO[7:0]<=~maxi;
                                        bbuff[0]<=~maxc;

bbuff[1]<=~count_out[indexo];
                                        end
                                //LED[2]<=1;
                                ARDUINO_INT<=1;
                                end
        if(repo==40)
                begin
                                ARDUINO_INT<=0;
                                SEL<=3'b101;
                                indexo<=indexo+1'b1;
                end
        if(repo==80)
                begin
                                ARDUINO_INT<=1;
                                ARDUINO_IO[7:0]<=bbuff[0];
                end
        if(repo==120)
                begin
                                ARDUINO_INT<=0;
                                SEL<=3'b110;
                end
        if(repo==160)
                begin
                                ARDUINO_INT<=1;
```

```
                    ARDUINO_IO[7:0]<=bbuff[1];
                end
            if(repo==200)
                begin
                    ARDUINO_INT<=0;
                    SEL<=3'b011;
                end
            end
end
endmodule
```

Halaman ini sengaja dikosongkan

LAMPIRAN IV
PROGRAM ITD PADA ARDUINO

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);
int serout=0;
int state=0;
int out=0;
unsigned int stoch=0;
int strep=0;
int lup=0;
bool Sel[3];
// the setup routine runs once when you press reset:
void setup() {
  lcd.init();                    // initialize the lcd
  // Print a message to the LCD.
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Posisi Suara :");
  // initialize serial communication at 9600 bits per second:
  attachInterrupt(0, count, RISING);
  //attachInterrupt(1, count_out_1, RISING);
  Serial.begin(115200);
  // init input:
  pinMode(5, INPUT);
  pinMode(6, INPUT);
  pinMode(7, INPUT);
  pinMode(8, INPUT);
  pinMode(9, INPUT);
  pinMode(10, INPUT);
  pinMode(11, INPUT);
  pinMode(12, INPUT);
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
```

```
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(12,HIGH);
  digitalWrite(11,HIGH);
  digitalWrite(10,HIGH);
  digitalWrite(9,HIGH);
  digitalWrite(8,HIGH);
  digitalWrite(7,HIGH);
  digitalWrite(6,HIGH);
  digitalWrite(5,HIGH);
  digitalWrite(A0,HIGH);
  digitalWrite(A1,HIGH);
  digitalWrite(A2,HIGH);
}

void stoc_out() {
  stoch=stoch<<8;
  if(digitalRead(12)==LOW)
     stoch=stoch|1;
   if(digitalRead(11)==LOW)
     stoch=stoch|2;
   if(digitalRead(10)==LOW)
     stoch=stoch|4;
   if(digitalRead(9)==LOW)
     stoch=stoch|8;
   if(digitalRead(8)==LOW)
     stoch=stoch|16;
   if(digitalRead(7)==LOW)
     stoch=stoch|32;
   if(digitalRead(6)==LOW)
     stoch=stoch|64;
   if(digitalRead(5)==LOW)
     stoch=stoch|128;
   strep++;
   if(strep>=2){
    Serial.println(stoch,BIN);
   // Serial.println();
```

```
   strep=0;
   }
}
void rst(){
  stoch=stoch<<32;
}

void count(){
  Sel[0]=digitalRead(A0);
  Sel[1]=digitalRead(A1);
  Sel[2]=digitalRead(A2);
  if(digitalRead(12)==LOW)
      serout=serout|1;
   if(digitalRead(11)==LOW)
      serout=serout|2;
   if(digitalRead(10)==LOW)
      serout=serout|4;
   if(digitalRead(9)==LOW)
      serout=serout|8;
   if(digitalRead(8)==LOW)
      serout=serout|16;
   if(digitalRead(7)==LOW)
      serout=serout|32;
   if(digitalRead(6)==LOW)
      serout=serout|64;
   if(digitalRead(5)==LOW)
      serout=serout|128;
      if(Sel[0]==LOW)
        count_out_0();
      if(Sel[1]==LOW)
        count_out_1();
      if(Sel[2]==LOW)
        count_out_2();
}
// the loop routine runs over and over again forever:
void count_out_0() {
```

```
  // read the input pin:
     // print out the result:
  Serial.print("hasil 0 ");
  Serial.print(serout);
  Serial.print("\t");
  serout=0;
  //delay(1);        // delay in between reads for stability
}

void count_out_1() {
  // read the input pin:
     // print out the result:
  Serial.print("hasil 1 ");
  Serial.print(serout);
  Serial.print("\t");
  serout=0;
  //delay(1);        // delay in between reads for stability
}

void count_out_2() {
  // read the input pin:
     // print out the result:
  Serial.print("hasil kali ");
  state=asin(serout*2.56*384/20);
  Serial.print(state);
  Serial.print("\t");
  if(state>100){
    Serial.println("Kanan");
    out=0;}
  else if(state<70){
    Serial.println("Kiri");
    out=1;}
  else if(state==90){
    Serial.println("Tengah");
    out=2;}
  serout=0;
```

```
  //delay(1);        // delay in between reads for stability
}

void loop(){
  lcd.setCursor(0,1);
    if(out==0)
    lcd.print("Kanan ");
    else if(out==1)
    lcd.print("Kiri  ");
    else if(out==2)
    lcd.print("Tengah");

  lcd.setCursor(8,1);
  lcd.print(state-90);
  lcd.print("  ");

}
```

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Penulis buku ini memiliki nama lengkap Rifqi Yunus Trisna Pratama, memiliki nama panggilan formal Rifqi dan beberapa nama panggilan lain dari setiap komunitas. Lahir pada Sabtu, 20 Desember 1997. Merupakan anak pertama dari tiga bersaudara. Sempat merasakan masa kecil di 3 kota berbeda yaitu Bojonegoro, Kendari dan Gresik. Memiliki kecenderungan untuk menemukan solusi 27 detik sebelum *deadline* serta suka bermain-main dengan benda-benda rusak dengan harapan dapat mempelajari cara kerja sekaligus memperbaikinya. Pendidikan pertamakali ditempuh di Kendari di TK Graha Asri. Selanjutnya bersekolah di dua sekolah dasar yaitu di SDN Randuagung IV dan SDN Sidokumpul II karena beberapa alasan. Setelah itu saya terdaftar sebagai siswa SMPN 1 Gresik terhitung dari tahun 2009 hingga 2012. Lalu penulis melanjutkan jenjang pendidikan di SMAN 1 Gresik hingga tahun 2015. Selanjutnya, penulis berhasil diterima di Departemen Teknik Elektro ITS. Penulis cukup sering berganti-ganti ketertarikan dan bergabung di beberapa bidang keminatan baik dalam bidang robotika, bahasa asing maupun teknologi.

Email : rifqiyunus@yahoo.co.id
HP/WA : 083831390016
Facebook : rifqi.yunus
Line : rifqi-yunus