BAB II

DASAR TEORI

2.1 Pengertian Sinyal

Sinyal merupakan suatu fungsi dengan satu variabel tak bebas dan satu atau beberapa variabel bebas. Sebuah sistem dapat didefinisikan sebagai sebuah alat fisik yang melakukan operasi pada suatu sinyal. Contohnya sebuah filter yang merupakan sistem yang dapat mengurangi derau (*noise*). Dalam hal ini filter melakukan beberapa operasi pada sinyal yang mempunyai efek mengurangi *noise* dari sinyal yang tercampur *noise*.

2.2 Pemrosesan Sinyal

Bila kita melewatkan sinyal melalui sebuah sistem, seperti pada pemfilteran, maka sinyal tersebut telah kita proses. Dalam pemrosesan sinyal–sinyal digital pada sebuah filter digital, operasi–operasi yang dilakukan pada sinyal terdiri dari sejumlah operasi matematis seperti yang ditetapkan oleh program perangkat lunak. Misalnya operasi penjumlah, pengurang, pembagi dan pengali.

Pemrosesan sinyal digital bisa dilakukan dengan mengubah sinyal analog menjadi sinyal digital, kemudian baru memproses sinyal tersebut. Setelah selesai kemudian diubah kembali menjadi sinyal analog. Pemrosesan sinyal dapat juga langsung dilakukan jika sinyal yang didapat sudah dalam bentuk digital.

2.3 Filter

Istilah filter sering digunakan sebagai alat baik berupa perangkat keras maupun perangkat lunak komputer yang diimplementasikan untuk mengolah sekumpulan data ber*noise* menjadi informasi yang diinginkan. *Noise* dapat diakibatkan dari berbagai sumber.

Filter secara umum dapat dibagi menjadi dua, yaitu filter analog dan filter digital. Filter analog dibagi menjadi dua yaitu filter aktif dan filter pasif. Filter pasif adalah filter yang menggunakan komponen pasif, seperti Resistor, Capasitor dan Induktor. Sedangkan filter aktif adalah filter yang menggunakan komponen aktif, seperti transistor dan IC. Filter digital termasuk dalam filter aktif karena menggunakan IC.

Filter digital secara strukturnya dibedakan menjadi dua yaitu filter digital *non recursive* dan filter digital *recursive*. Filter digital *non recursive* adalah filter yang outputnya hanya tergantung pada input. Sedangkan filter digital *recursive* adalah filter yang outputnya tergantung dari input dan keluaran sebelumnya.

2.3.1 Filter Wiener

Untuk menghasilkan filter digital yang optimum, salah satu pendekatan yang sering digunakan adalah filter Wiener. Perancangan filter Wiener memerlukan pengetahuan yang tinggi tentang data yang harus diproses. Filter hanya akan optimum bila karakteristik statistis data benar-benar diketahui dan cocok dengan anggapan yang mendasari perancangan filter. Pada kenyataannya sangatlah sulit untuk mendapatkan karakteristik statistis data yang tepat. Untuk itu

diperlukan suatu filter yang memiliki kemampuan untuk beradaptasi sendiri yang kemudian mendasari ide perancangan suatu filter adaptif.

Untuk mengoptimalkan bobot, filter wiener menggunakan persamaan

$$W^* = R^{-1} P \tag{2.1}$$

dimana R = matriks autokorelasi antara d (*desired respone*)

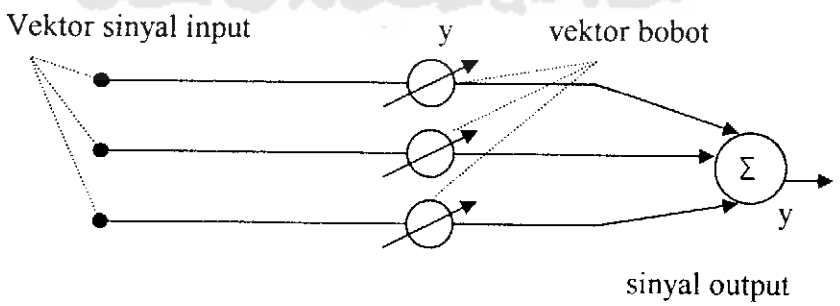
P = matriks korelasi silang antara d dan u (input)

W* = bobot optimum

Untuk mencapai bobot optimum, diperlukan suatu algoritma adaptif. Adaptasi pada filter dimaksudkan untuk mendapatkan penyelesaian bobot optimal. Salah satu algoritma yang dikenal adalah algoritma *Recursive Least Square* (RLS). Algoritma ini mempunyai perhitungan matematis yang cukup kompleks.

2.3.2 Filter Adaptif

Filter adaptif merupakan dasar dari *Adaptive Signal Prosessing*. Filter adaptif lebih mudah dipelajari dan dianalisa. Filter Adaptif ini sesuai untuk filter digital kerena pembentukannya yang sederhana. Secara umum diagram filter adaptif ditunjukkan pada gambar sebagai berikut :



Gambar 2.1 Bentuk Umum dari Filter Adaptif

Aplikasi filter adaptif sangat bermacam-macam. Beberapa konfigurasi kadangkala merupakan kombinasi dari beberapa konfigurasi dasar. Meskipun berbeda dalam berbagai aplikasi, tetapi pada umumnya memiliki bagian yang sama, yaitu vektor masukan dan respon yang diinginkan digunakan untuk mengoptimalkan kesalahan dan untuk mengendalikan nilai sekumpulan koefisien filter yang diukur. Koefisien terukur dapat berupa bobot, koefisien refleksi atau parameter rotasi tergantung struktur filter tersebut. Dalam sistem adaptasi ada 4 macam konfigurasi, yaitu:

1. Prediksi (*Prediction*)

   Sistem adaptif pada gambar 2.2 adalah filter adaptif yang digunakan untuk memprediksi nilai y supaya sama dengan d, sehingga diperoleh nilai galat e yang minimal atau nol. Secara umum jenis filter ini dalam penerapannya digunakan untuk pengurangan *noise*.

2. Identifikasi Sistem (*System Identification*)

   Sistem adaptif pada gambar 2.3 adalah sebuah filter adaptif yang digunakan sebagai model linier dan menghasilkan bentuk terbaik untuk *plant* yang tidak diketahui.

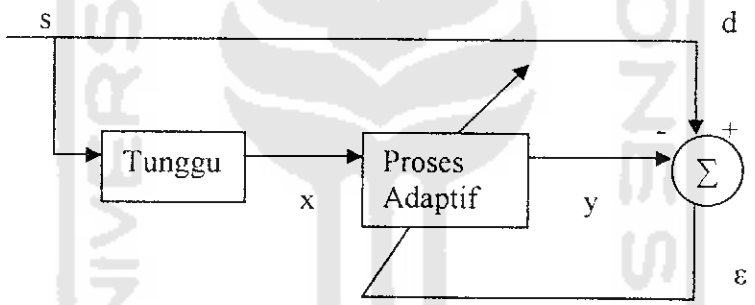
3. Pemodelan Invers (*Inverse Modeling*)

   Sistem adaptif pada gambar 2.4 adalah filter adaptif yang berfungsi untuk memproses sinyal masukan (s) yang tertunda dengan panjang tundanya disesuaikan dengan proses yang berlangsung pada *plant* dan sudah terakumulasi dengan galat yang ada, bisa diperoleh galat yang digunakan

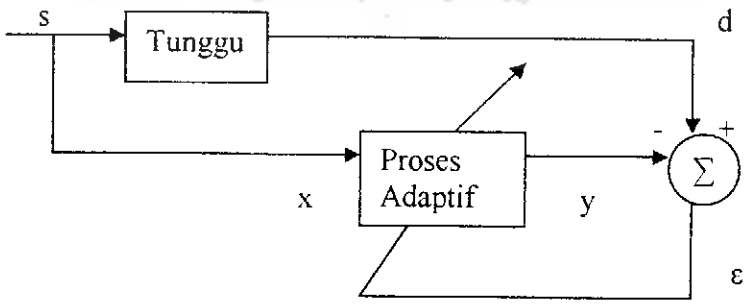
untuk mengukur filter adaptifnya sampai diperoleh keluaran yang diinginkan.

4. Penghapus Derau (*Noise Canceler*)

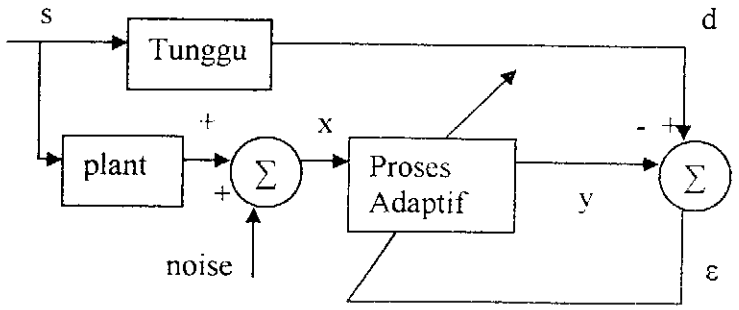
Sistem adaptif pada gambar 2.5 adalah filter adaptif yang berfungsi untuk menghapus *noise*. Disini kita memiliki sinyal S yang telah terakumulasi dengan *noise* n. *Noise* ini memiliki korelasi dengan *noise* lainnya n' yang tidak lain merupakan masukan yang nantinya akan diproses oleh filter adaptif. Filter adaptif berfungsi untuk mengolah sinyal n' menjadi sama dengan n, sehingga $\varepsilon \approx S$.



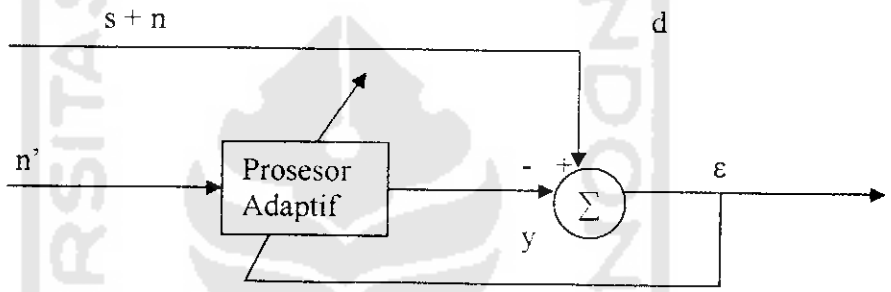
Gambar 2.2 Sistem Adaptif untuk Prediksi (*Prediction*)



Gambar 2.3 Sistem Adaptif untuk Identifikasi Sistem (*System Identification*)



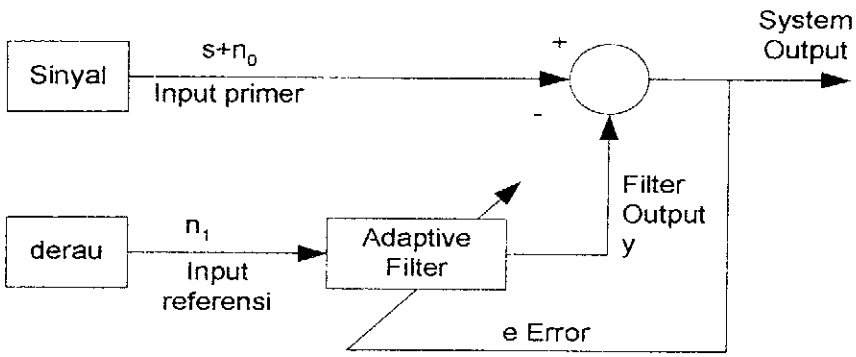
Gambar 2.4 Sistem Adaptif untuk Pemodelan Invers (*Inverse Modeling*)



Gambar 2.5 Sistem Adaptif untuk Penghilang Derau (*Interference Canceler*)

2.4 Sistem Penghapus Derau (*Noise Canceler*)

Pada bagian ini akan dibahas tentang suatu sistem filter adaptif yang banyak digunakan dalam pemrosesan sinyal dan pengaturan bobot-bobot pada setiap perulangan program yang dijalankan. Jika terdapat *noise* pada sistem dalam masukan maka filter adaptif ini harus bisa memanipulasi setiap masukan supaya menghasilkan sinyal yang kita inginkan, tanpa terpengaruh dengan *noise* yang ada. Adapun diagram sistem penghapus *noise* adaptif ini ditunjukkan pada gambar 2.6:



Gambar 2.6 Diagram Sistem Penghapus *Noise* Adaptif

Prinsip kerja dari sistem ini adalah dengan mengurangi *noise* dari sinyal yang diterima. Sistem ini menggunakan dua masukan yang diperoleh dari dua input yaitu input primer / utama dan input referensi / acuan.

Input utama (s) yang sudah tercampur dengan *noise* (n_0) disebut juga sinyal yang diinginkan (*desired signal*, d_k), dengan persamaan sebagai berikut :

$$d_k = s + n_0 \tag{2.2}$$

Sinyal s dan n_0 tidak berkorelasi, tetapi sinyal n_0 berkorelasi dengan sinyal n_1. Sinyal acuan n_1 (nantinya akan dianggap x) akan diproses oleh filter adaptif dalam hal ini menggunakan jenis algoritma RLS (*Recursive Least Square*) untuk menghasilkan sinyal keluaran

$$y_k = w_k.x_k \tag{2.3}$$

Dan w_k merupakan bobot-bobot real yang bisa diatur (*adjustable*) melalui filter adaptif. Keluaran filter (y_k) ini akan diperoleh selisihnya dengan hasil sinyal yang diperoleh input utama, berupa sinyal yang diinginkan (d_k). Selisih ini disebut juga dengan kesalahan (*error*) dengan hubungan sebagai berikut:

$$\varepsilon_k = d_k - y_k \tag{2.4}$$

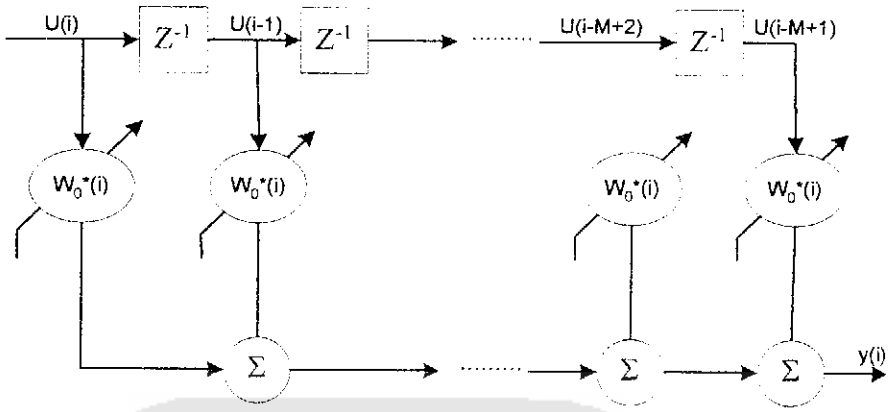
Sinyal kesalahan inilah yang akan digunakan untuk mengatur (*adjust*) nilai dari bobot-bobot w_k dengan menggunakan algoritma RLS.

Pada masalah ini filter adaptif akan 'mematikan dirinya sendiri' dan menghasilkan nilai 0 untuk keluaran y_k, dengan begitu sistem penghapus *noise* adaptif tidak akan mempengaruhi sama sekali nilai sinyal yang diinginkan (d_k).

2.5 Algoritma *Recursive Least Square* (RLS)

Algoritma ini merupakan algoritma *Least Square* yang bersifat rekursif (perulangan). Mirip dengan algoritma *Least Mean Square* (LMS) yang mengestimasikan nilai bobot yang sudah diperbaharui, algoritma *Recursive Least Square* (RLS) dengan estimasi kuadrat terkecil (*least square*) dari vektor bobot saat n-1 bisa menghitung vektor bobot terestimasi saat n dengan adanya data baru.

Kemampuan khusus yang membedakan dari LMS adalah algoritma ini memiliki kemampuan untuk mengggunakan seluruh informasi pada data masukan dengan memperluasnya sampai ke masukan yang sebenarnya hingga saat inisialisasi dilakukan. Kemampuan inilah yang menyebabkan algoritma RLS memiliki kecepatan konvergensi yang lebih tinggi dibandingkan dengan algoritma LMS. Konsekuensinya algoritma RLS memiliki beban komputasi yang lebih besar, karena perhitungan matematis yang lebih kompleks dibanding algoritma LMS. Gambar 2.7 menunjukkan struktur filter transversal dengan sumber algoritma RLS.



Gambar 2.7 Filter Transversal

Gambar 2.7 merupakan struktur filter transversal yang menggambarkan pemodelan dari algoritma RLS yang dibedakan notasinya dari algoritma LMS dengan vektor masukan tersadapnya saat i adalah

$$u(i) = [u(i), u(i-1), ..., u(i-M+1)]^T \tag{2.5}$$

dan w(n) adalah vektor bobot tersadap saat n, yang didefinisikan sebagai

$$w(n) = [w_o(n), w_1(n), ..., w_{M-1}(n)]^T \tag{2.6}$$

Fungsi harga yang akan diminimisasi ε(n) sering disebut juga faktor pembobotan yang didefinisikan sebagai

$$\varepsilon(n) = \sum_{i=1}^{n} \lambda^{n-1} |e(i)|^2 \tag{2.7}$$

dengan λ adalah konstanta positif yang bernilai kurang dari 1. Jika nilainya sama dengan 1 maka akan terlihat metode kuadrat terkecil yang asli. Nilai invers dari 1-λ menunjukkan ukuran memori algoritma, untuk kasus khusus dengan λ=1 berarti memorinya tidak terbatas (*infinite memory*). Nilai ini jika dibandingkan dengan nilai parameter LMS akan dipakai hubungannya.

$$\lambda = 1 - \mu \tag{2.8}$$

Dan e(i) adalah perbedaan antara respons yang diinginkan d(i) dan keluaran y(i) yang dihasilkan oleh filter transversal, dilanjutkan oleh hubungan:

$$\begin{aligned} e(i) &= d(i) - y(i) \\ &= d(i) - w^T(n).u(n) \end{aligned} \tag{2.9}$$

Berdasarkan persamaan (2.9) maka diperoleh kesimpulan bahwa nilai e(i) atau galat merupakan fungsi dari respons yang diinginkan dikurangi hasil perkalian vektor bobot dan vektor masukan tersadapnya.

Nilai galat inilah yang nantinya digunakan untuk memperbaharui bobot-bobot baru yang akan menentukan kecepatan konvergensi dari algoritma RLS ini. Jika galat estimasi sudah mencapai nol maka sistem akan mempertahankan nilai bobot yang paling terakhir, w*(n). Vektor bobot inilah yang disebut penyelesaian filter wiener.

Nilai optimum dari vektor bobot tersadap w*(n) didefinisikan oleh persamaan normal yang ditulis dalam bentuk matriks:

$$\Phi(n)w^*(n) = \theta(n) \tag{2.10}$$

$\Phi(n)$ adalah matriks korelasi berdimensi MxM yang adalah

$$\Phi(n) = \sum_{i=1}^{n} \lambda^{n-1} u(i) u^T(i) \tag{2.11}$$

Sedangkan matriks korelasi silang $\theta(n)$ berdimensi Mx1 antara masukan tersadap dan respons yang diinginkan, didefinisikan sebagai

$$\begin{aligned} \theta(n) &= \sum_{i=1}^{n} \lambda^{n-1} u(i) d(i) \\ &= \lambda\theta(n-1) + u(n)d(n) \end{aligned} \tag{2.12}$$

Pembaharuan bobot algoritma RLS diberikan oleh persamaan

$$w(n) = w(n-1) + k(n)\alpha(n) \tag{2.13}$$

dengan k(n) atau vektor korelasi perolehannya bernilai

$$k(n) = \frac{\lambda^{-1}P(n-1)u(n)}{1+\lambda^{-1}u^{H}(n)P(n-1)u(n)} \tag{2.14}$$

dan

$$\alpha(n) = d(n) - w^{H}(n-1)u(n) \tag{2.15}$$

dimana perlu diperhatikan bahwa proses pembaharuan bobot memori tergantung pada persamaan (2.15) yang merupakan galat terdahulu, namun untuk mencari nilai minimal dari fungsi harga di atas dipengaruhi oleh galat berikutnya,

$$e(n) = d(n) - w^{T}(n)u(n) \tag{2.16}$$

Kedua macam galat di atas memiliki persamaan yaitu merupakan persamaan estimasi kesalahan dari sistem. Perbedaannya ialah $\alpha(n)$ adalah nilai estimasi kesalahan saat vektor bobot belum diperbahuri / estimasi kesalahan saat sekarang (*prior estimation error*), sedangkan e(n) adalah estimasi kesalahan saat vektor bobot sudah diperbaharui (*posteriori estimation error*).

Untuk persamaan (2.14) nilai P(n) ditentukan oleh persamaan matematis sebagai berikut:

$$P(n) = \lambda^{-1}P(n-1) - \lambda^{-1}k(n)u^{H}(n)P(n-1) \tag{2.17}$$

Dari persamaan-persamaan di atas terlihat adanya perbedaan yang sangat mencolok antara algoritma LMS dan RLS dalam bentuk aljabarnya, dimana algoritma RLS bersifat kompleks secara persamaan matematisnya, sedangkan algoritma LMS masih sederhana dalam persamaan matematisnya.

2.6. *Field Programmable Gate Array* (FPGA)

Field Programmable Gate Array merupakan sebuah piranti digital yang dapat diprogram untuk merepresentasikan sistem logika yang telah dirancang. Teknologi IC FPGA diperkenalkan pada tahun 1985 oleh perusahaan semikonduktor Xilinx. FPGA adalah sebuah konsep teknologi IC yang dapat diprogram dan dihapus seperti halnya RAM. FPGA kemudian berkembang pesat, baik dari segi kepadatan gerbang, kecepatan dan disertai dengan penurunan harga jual.

Penemuan FPGA telah membuat peningkatan yang pesat akan pembuatan prototipe beberapa sistem digital. Salah satu produsen FPGA yang ada dipasaran adalah Xilinx, disamping produsen lainnya Actel dan Altera. Prinsip dasar dari pemrograman / pengkonfigurasian FPGA Xilinx adalah pengubahan gambar rangkaian elektronik digital dari perangkat lunak Xilinx menjadi file aliran bit (*bitstream*) dan dikonfigurasi (*didownload*) ke dalam IC FPGA Xilinx tersebut sehingga IC tersebut terkonfigurasi secara perangkat keras yang dirancang dalam perangkat lunak Xilinx. FPGA produk Xilinx sudah melewati beberapa generasi antara lain XC2000, XC3000 dan XC4000. Tiap generasi memiliki sifat dan kemampuan yang berbeda. Perbedaan tersebut meliputi kecepatan, kapasitas gerbang logika, jumlah CLB dan jumlah IOB.

2.6.1 FPGA keluarga Xilinx Spartan II

Spartan II merupakan salah satu keluarga FPGA yang dikeluarkan oleh Xilinx. Xilinx merupakan salah satu pabrik pembuat FPGA yang cukup terkenal. Keluarga Spartan II merupakan keluarga FPGA yang memiliki 15.000 sampai

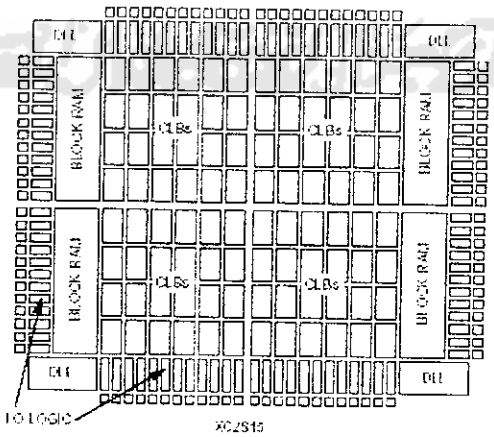
200.000 gerbang. IC Xilinx ini dapat diprogram dan dihapus dengan waktu yang tidak terbatas serta murah harganya. Kelurga Spartan ini dapat diprogram dengan mudah menggunakan Xilinx *Development System* ataupun dengan *development system* lain yang dikembangkan oleh para pengguna.

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O[1]	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	132	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	196	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

Tabel 2.1 Data Keluarga Spartan II

2.6.2 Struktur Dasar Keluarga Spartan II

Suatu piranti FPGA terdiri atas *Configurable Logic Blocks* (CLB), unit input/output, empat buah *Delay-Locked Loops* (DLLs), unit RAM dan unit routing yang dapat diprogram secara otomatis penuh. Susunan dan letak masing-masing bagian tersebut dapat dilihat pada gambar 2.8.

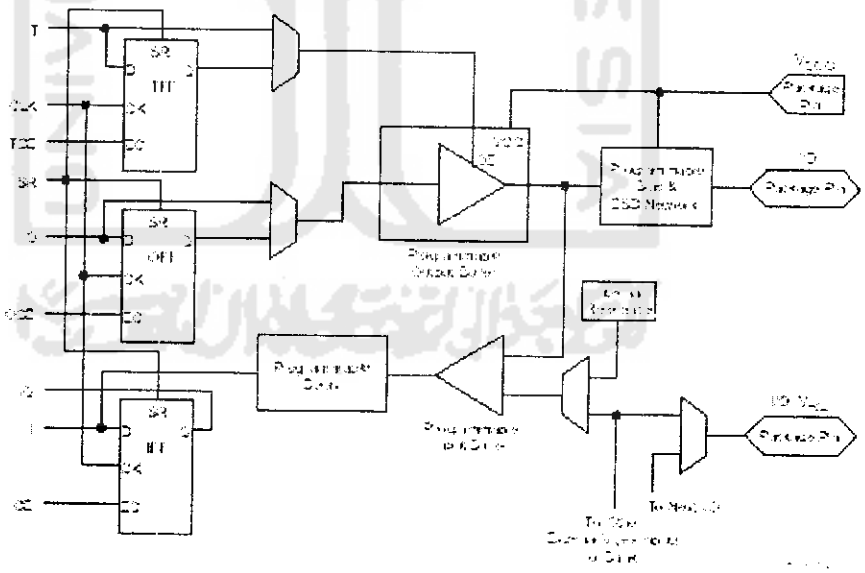


Gambar 2.8. Diagram Blok Dasar Keluarga Spartan II

2.6.2.1 *Input/Output Blocks* (IOB)

Input/Output blocks merupakan bagian dari FPGA yang berfungsi menghubungkan FPGA dengan piranti lain yang akan terkoneksi IOB keluarga Spartan II mampu bekerja pada berbagai macam standar I/O seperti TTL, CMOS dan PCI. Kemampuan untuk menyesuaikan dengan berbagai macam I/O didukung dengan kemampuan tiap *pad* I/O untuk ditambahi *pull-up* dan *pull-down resistor.*

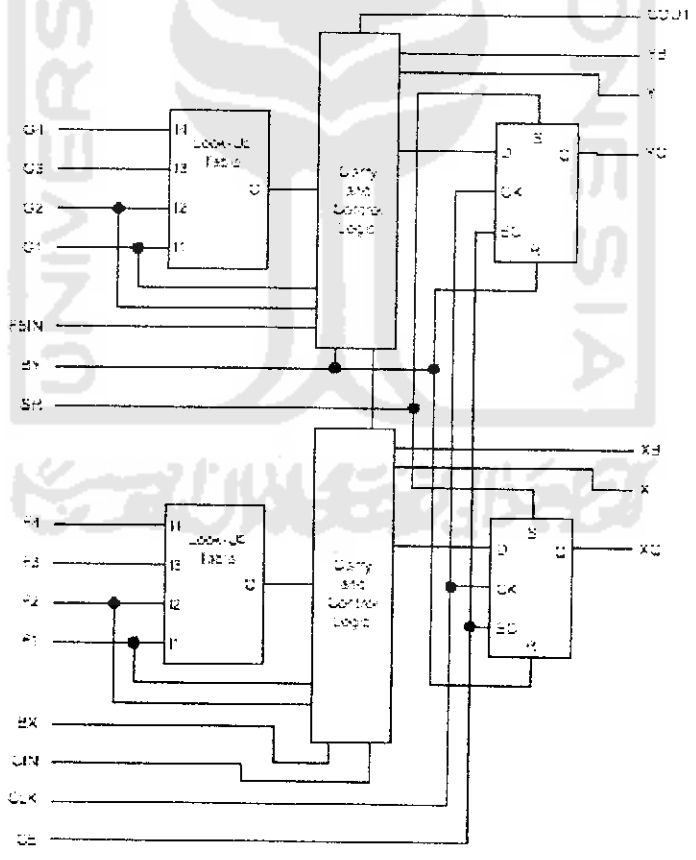
Bagian *buffer* pada Spartan II IOB *input path* akan menghubungkan sinyal input yang masuk secara langsung dengan logika internal atau secara tidak langsung melalui input flip-flop optional. Sedangkan bagian *output path* termasuk buffer 3 keadaan akan men-*drive* sinyal output menuju pad output. Sama dengan sinyal input, sinyal output ini dapat dihubungkan dengan pad output melalui logika internal maupun melalui output flip-flop optional.



Gambar 2.8 Blok IOB Spartan II

2.6.2.2 *Configurable Logic Blocks* (CLB)

CLB merupakan bagian dari FPGA yang berfungsi merubah logika-logika terprogram yang dimasukkan menjadi fungsi-fungsi yang dipahami oleh FPGA dan dapat bekerja sesuai dengan program yang diinginkan. CLB Spartan II terdiri atas *Logic Cell (LC)* sebagai bangunan utama. Sebuah LC terditi atas 4 buah input yang akan membangkitkan fungsi logika yang diinginkan, *carry logic* dan elemen penyimpan. Keluaran setiap LC akan men-*drive* keluaran CLB dan input pada D flip-flop. Setiap CLB pada Spartan II terdiri atas empat LC yang tersusun dalam dua *slices* yang identik. Tiap CLB ini juga memuat logika yang akan mengkombinasi generator pembangkit fungsi logika untuk 4 sampai 6 input.



Gambar 2.10 CLB pada Spartan II

Generator pembangkit fungsi diimplementasikan dalam sebuah *look-up tables* (LUT) 4 input. LUT ini juga dapat membangkitkan sebuah RAM 16 x 1 sinkron serta membangkitkan fungsi *shift register* 16 bit. Fungsi RAM yang dibangkitkan generator ini akan melengkapi *block RAM* yang dimiliki oleh Spartan II sehingga mampu menghasilkan unit penyimpan data yang handal.

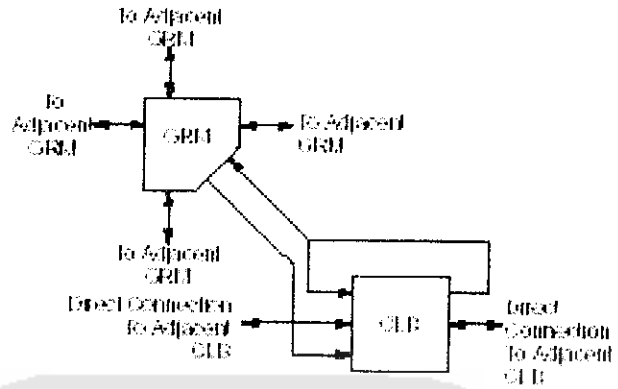
2.6.2.3 *Programmable Routing Matrix*

Programmable Routing Matrix merupakan cara sebuah FPGA melakukan *routing* menghubungkan CLB-CLB dan IOB-IOB yang digunakan dalam desain menjadi satu kesatuan sistem. *Routing* ini dilakukan secara otomatis penuh. Namun untuk keperluan tertentu, optimasi jalur yang paling pendek dapat dilakukan *routing* manual.

Dalam keluarga Spartan II ada beberapa macam routing yang bisa digunakan yaitu:

1. *Local Routing*

   Local Routing digunakan untuk mengimplementasikan hubungan antara LUT, flip-flop dan *General Routing Matrix* (GRM), antara jalur umpan balik internal CLB dengan LUT lain pada CLB yang sama untuk koneksi *high speed,* serta antara jalur-jalur langsung yang bisa dibuat untuk memperkecil *delay.*



Gambar 2.11 Struktur *Local Routing*

2. *General Purpose Routing*

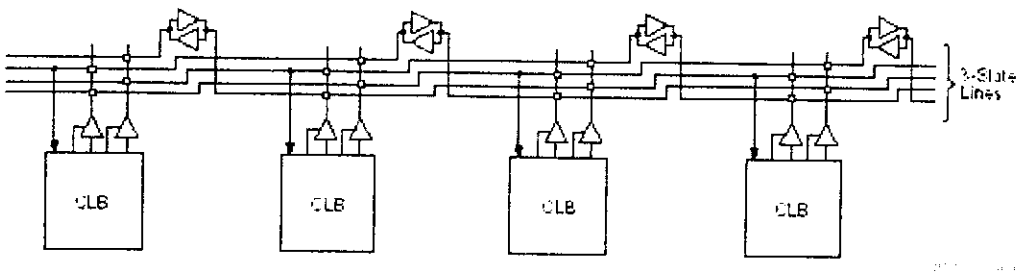
   General Purpose Routing digunakan untuk melakukan koneksi vertikal dan horizonal antar kolom dan baris CLB yang digunakan.

3. *I/O Routing*

   I/O Routing khusus digunakan untuk menghubungkan *array* pada CLB dengan IOB.

4. *Dedicated Routing*

   Dedicated Routing digunakan untuk menguhubungkan beberapa CLB, IOB ataupun LUT yang memerlukan perlakuan khusus untuk memaksimalkan performa.



Gambar 2.12 Koneksi BUFT untuk *dedicated Horizontal Bus Line*

5. *Global Routing*

   Global Routing digunakan untuk menghubungkan clock dengan bagian yang membutuhkan serta sinyal-sinyal dengan *fan-out* yang tinggi ke bagian lain.

2.6.3 Pemrograman FPGA

Keluarga Spartan II yang dikeluarkan oleh Xilinx dapat dengan mudah diprogram menggunakan *development system software* keluaran Xilinx yaitu Xilinx Foundation Series. Saat ini versi yang terbaru adalah versi 6.i3. *Software* memiliki kemampuan *place and route tools* (PAR) yang memungkinkan penggunanya menyusun IOB, LUT dan CLB menjadi sebuah kesatuan sistem.

FPGA dapat diprogram menggunakan metode *schematic* ataupun menggunakan *Hardware Description Language*, baik Verilog maupun VHDL. Masing-masing metode memiliki kelebihan dan kelemahannya sendiri.

2.6.4 Mode Operasi

Keluarga Spartan II dapat dioperasikan dalam 4 mode yaitu:

1. *Slave Serial mode*
2. *Master Serial mode*
3. *Slave Parallel mode*
4. *Boundary Scan mode*

Mode yang paling mudah digunakan adalah *Boudary Scan Mode* dimana tidak diperlukan koneksi-koneksi khusus, cukup menggunakan kabel paralel yang dikoneksikan menggunakan JTAG, maka desain dapat dengan mudah diimplementasikan ke dalam FPGA.

A & E connector		
Pin	Signal	S-II pin
1	GND	-
2	VU	-
3	VDD33	-
4	A4	70
5	A5	69
6	A6	68
7	A7	67
8	A8	63
9	A9	62
10	A10	61
11	A11	60
12	A12	59
13	A13	58
14	A14	57
15	A15	49
16	A16	48
17	A17	47
18	A18	46
19	A19	45
20	A20	44
21	A21	43
22	A22	42
23	A23	41
24	A24	37
25	A25	36
26	A26	35
27	A27	34
28	A28	33
29	A29	31
30	A30	30
31	A31	29
32	A32	27
33	A33	24
34	A34	23
35	A35	22
36	A36	21
37	A37	20
38	A38	18
39	A39	17
40	A40	16

B & F connector		
Pin	Signal	S-II pin
1	GND	-
2	VU	-
3	VDD33	-
4	B4	194
5	B5	193
6	B6	192
7	B7	191
8	B8	189
9	B9	188
10	B10	187
11	B11*	185
12	B12*	182
13	B13	-
14	B14	-
15	B15	-
16	B16	-
17	B17	-
18	NC	-
19	NC	-
20	NC	-
21	NC	-
22	NC	-
23	NC	-
24	NC	-
25	NC	-
26	NC	-
27	NC	-
28	NC	-
29	NC	-
30	NC	-
31	NC	-
32	NC	-
33	NC	-
34	NC	-
35	NC	-
36	NC	-
37	NC	-
38	NC	-
39	NC	-
40	NC	-

C connector		
Pin	Signal	S-II pin
1	GND	-
2	VU	-
3	VDD33	-
4	C4	181
5	C5	180
6	C6	179
7	C7	178
8	C8	176
9	C9	175
10	C10	174
11	C11	173
12	C12	172
13	C13	168
14	C14	167
15	C15	166
16	C16	165
17	C17	164
18	C18	163
19	C19	162
20	C20	161
21	C21	160
22	C22	154
23	C23	152
24	C24	151
25	C25	150
26	C26	149
27	C27	148
28	C28	147
29	C29	146
30	C30	142
31	C31	141
32	C32	140
33	C33	139
34	C34	138
35	C35	136
36	C36	135
37	C37	134
38	C38	133
39	C39	132
40	C40	129

D connector		
Pin	Signal	S-II pin
1	GND	-
2	VU	-
3	VDD33	-
4	D4	127
5	D5	125
6	D6	126
7	D7	122
8	D8	123
9	D9	120
10	D10	121
11	D11	115
12	D12	119
13	D13	113
14	D14	114
15	D15	111
16	D16	112
17	D17	109
18	D18	110
19	D19	102
20	D20	108
21	D21	100
22	D22	101
23	D23	98
24	D24	99
25	D25	96
26	D26	97
27	D27	94
28	D28	95
29	D29	89
30	D30	90
31	D31	87
32	D32	88
33	D33	84
34	D34	86
35	D35	82
36	D36	83
37	D37	75
38	D38	81
39	D39	73
40	D40	74

Tabel 2.1 Konfigurasi Pin yang Terdapat pada Rangkaian D2.