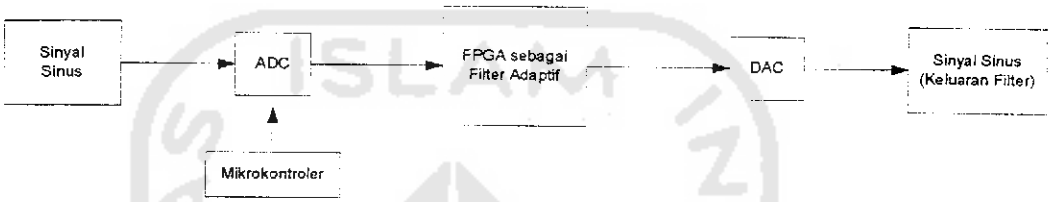
BAB III

PERANCANGAN SISTEM

Perancangan sistem secara garis besar diperlihatkan pada gambar 3.1 dibawah ini:



Gambar 3.1 Diagram Blok Sistem

Proses awal dalam diagram blok adalah perekaman suara. Suara yang sudah direkam, baik suara yang ber*noise* ataupun *noise* referensi, dimasukkan ke ADC (*Analog to Digital Converter*). Pada ADC data diubah dari analog ke data digital. Setelah data menjadi data digital, baru dapat diproses menggunakan FPGA xilinx. Keluaran dari FPGA masuk ke DAC (*Digital to Analog Converter*) untuk diproses kembali menjadi data analog. Setelah itu suara baru dikeluarkan melalui *speaker*.

3.1 Sistem Kerja

Filter adaptif merupakan filter digital yang secara dapat otomatis mengatur parameternya sendiri. Filter adaptif memerlukan dua masukan yaitu isyarat masukan x_k dan isyarat referensi d_k. Filter ini mempunyai kemampuan untuk memperbaharui koefisiennya sendiri. Koefisien yang baru kemudian dikirim ke

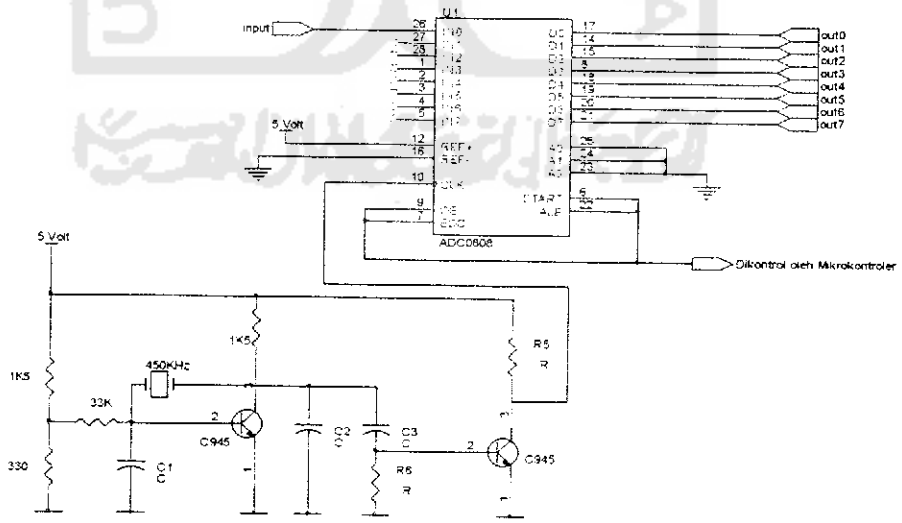
filter dari pembangkit koefisien. Pembangkit koefisien merupakan algoritma adaptif yang memodifikasi koefisien sebagai tanggapan terhadap isyarat masukan.

Keluaran filter adaptif y_k dibandingkan dengan isyarat referensi d_k untuk menghasilkan isyarat galat ε_k. Galat ini kemudian digunakan sebagai masukan ke algoritma RLS (*Recusive Least Square*) yang mengkorelasi bobot dari filter. Proses ini diulangi terus menerus sampai isyarat galat menjadi sangat kecil.

3.2 Perancangan Perangkat Keras

3.2.1 *Analog to Digital Converter* (ADC)

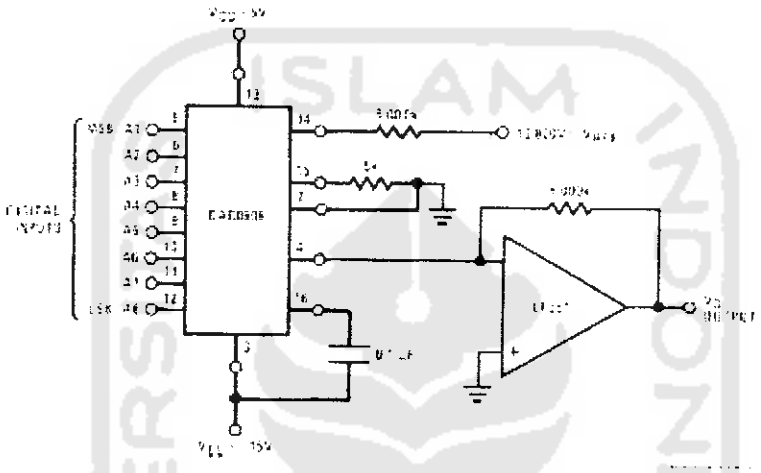
Perancangan rangkaian ADC pada penelitian ini menggunakan IC ADC 0808. IC ini mempunyai kaki-kaki kontrol yang tidak bisa dikontrol secara manual. Jadi untuk kaki-kaki tersebut dikontrol dengan menggunakan mikrokontroler AT89C51. Kaki-kaki tersebut adalah EOC (*End of Conversion*), OE (*Output Enable*), start dan ALE. IC ini mempunyai waktu konversi 100µs. Gambar 3.2 adalah rangkaian ADC yang menggunakan IC ADC 0808.



Gambar 3.2 Rangkaian *Analog to Digital Converter* (ADC)

3.2.2 *Digital to Analog Converter* (DAC)

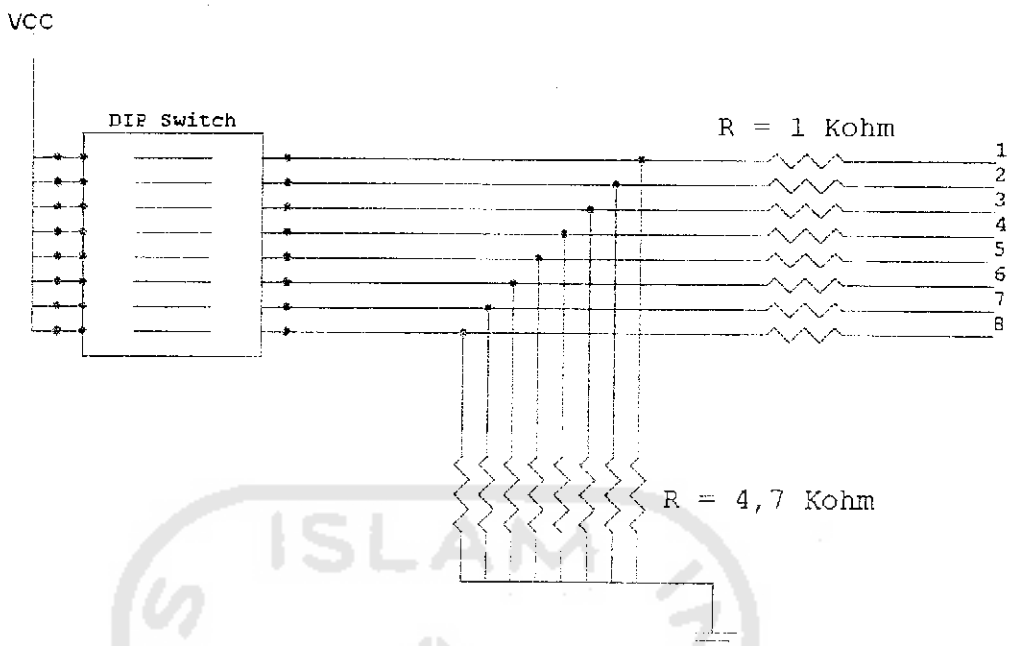
Untuk perancangan rangkaian DAC, pada penelitian ini menggunakan IC DAC 0808. IC ini mempunyai 8 bit masukan dan 1 keluaran. IC ini bertugas mengubah data digital menjadi data analog. Rangkaian ini membutuhkan sumber tegangan -15V, 15V, 10V dan 5V. Gambar 3.3 adalah rangkaian DAC 0808.



Gambar 3.3 Rangkaian *Digital to Analog Converter* (DAC)

3.2.3 Rangkaian DIP *Switch*

Rangkaian DIP *switch* ini berfungsi sebagai input tambahan yang digunakan untuk memberi tegangan ke preset, switch dan clear. Rangkaian ini menggunakan sebuah saklar DIP 8 jalur sebagai representasi sinyal input digital 8 bit. Rangkaian ini juga menggunakan hambatan (R) = 1 k ohm dan 4,7 k ohm. Dibawah ini adalah gambar rangkaian dari DIP switch.



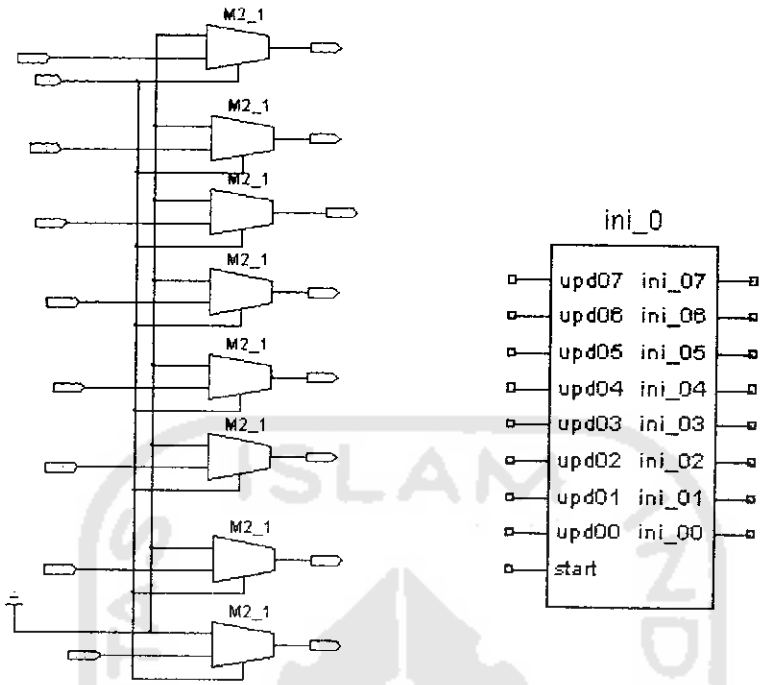
Gambar 3.4 Rangkaian DIP *Switch*

3.3 Perancangan Perangkat Lunak

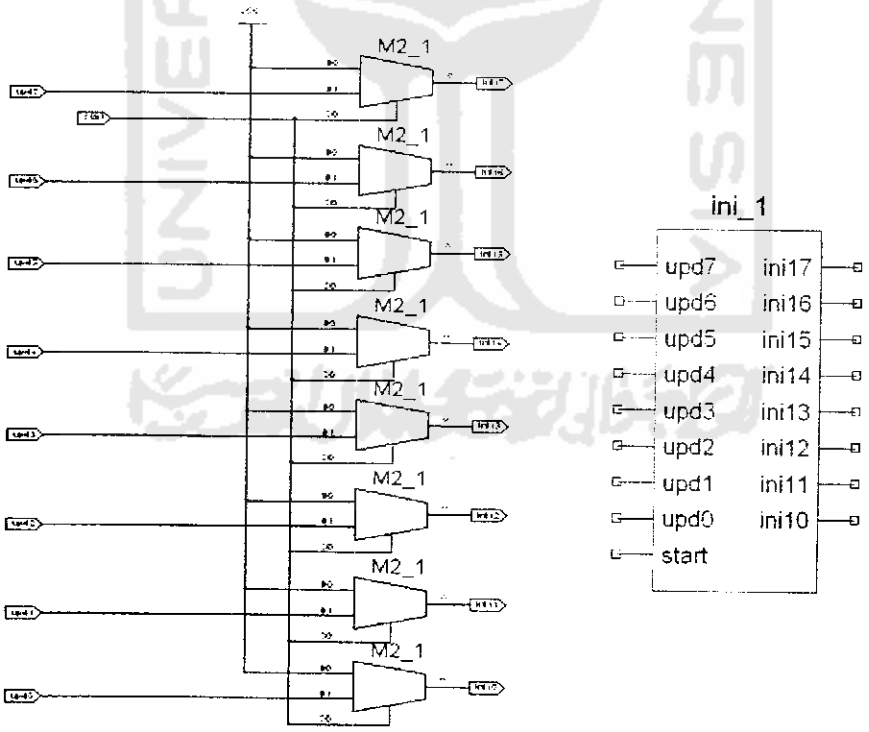
3.3.1 Perancangan Rangkaian Dasar

3.3.1.1 Rangkaian Inisialisasi

Pada perancangan filter adaptif dengan algoritma *Recursive Least Square* (RLS) membutuhkan proses inisialisasi untuk menjalankan sistem tersebut. Untuk menjalankan sistem filter adaptif ini, parameter bobot (w) dan matriks korelasi (P) memerlukan proses inisialisasi ini. Untuk bobot diset w_0=0, dan untuk $P_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ Untuk membuat rangkaian inisialisasi digunakan 8 buah mux 2_1. Gambar 3.5 dan 3.6 adalah rangkaian digital untuk nilai 0 dan 1.



Gambar 3.5 Rangkaian dan Blok Digital untuk Nilai Inisial = 0

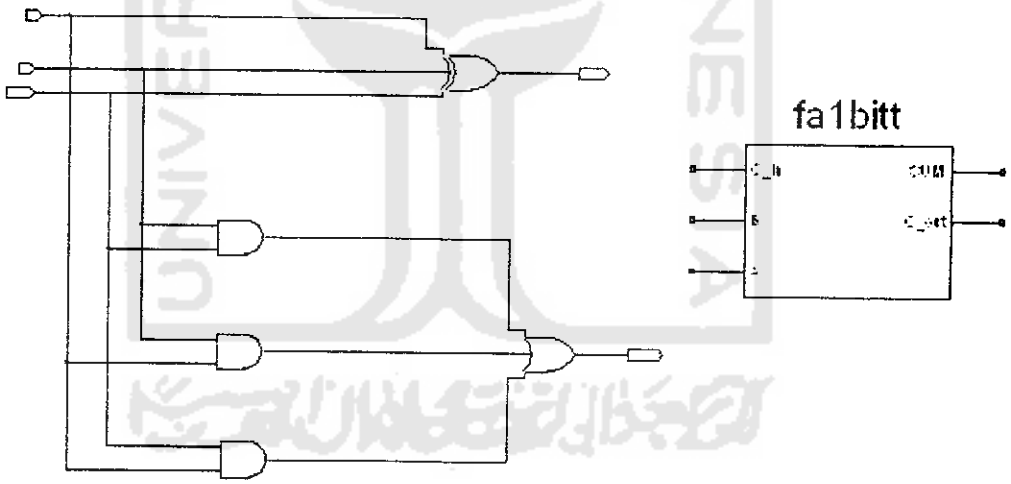


Gambar 3.6 Rangkaian dan Blok Digital untuk Nilai Inisial = 1

3.3.1.2 Rangkaian Penjumlah (*Full Adder*)

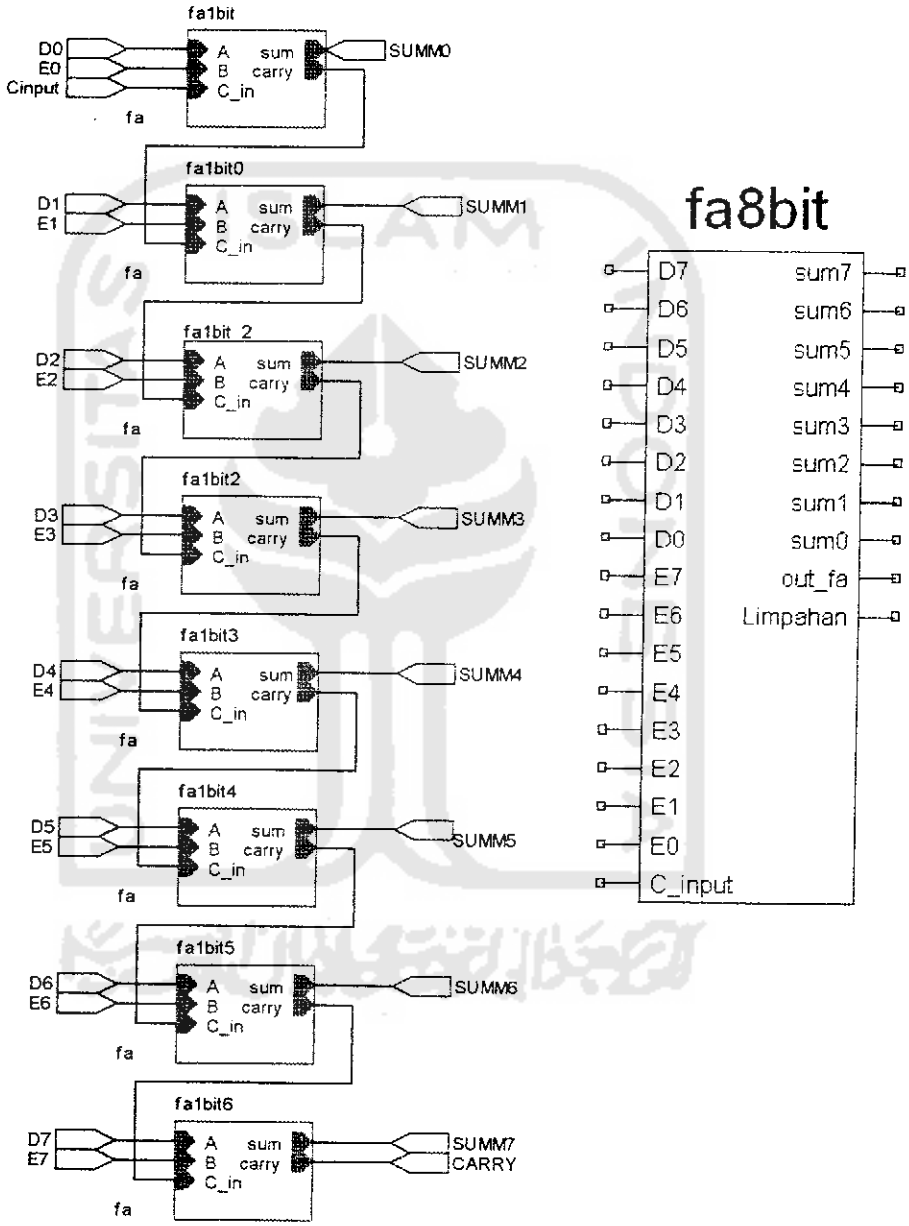
Proses penjumlahan bilangan biner sama seperti pada bilangan desimal yaitu dimulai dengan menjumlahkan LSB (*Least Significant Bit*) dari kedua bilangan masukan yang ingin dijumlahkan. Jadi 1+1 = 10, berarti penjumlahan untuk posisi ini adalah 0 dengan *carry* 1. *Carry* ini harus dijumlahkan ke posisi berikutnya bersamaan dengan kedua masukan pada posisi itu. Begitu seterusnya sampai posisi bit terakhir yaitu bit tanda.

Rangkaian *full adder* satu bit dibuat dengan menggunakan tiga gerbang AND2, satu gerbang OR3 dan XOR3. Input yang digunakan adalah A, B dan C_in. Sedangkan outputnya adalah SUM dan C_out.



Gambar 3.7 Rangkaian dan Blok *Full Adder* Satu Bit

Untuk membuat rangkaian *full adder* 8 bit menggunakan rangkaian *full adder* 1 bit sebanyak delapan buah. Rangkaian ini ditunjukkan pada gambar 3.8. Untuk rangkaian *full adder* 8 bit C_in diberi nilai 0.



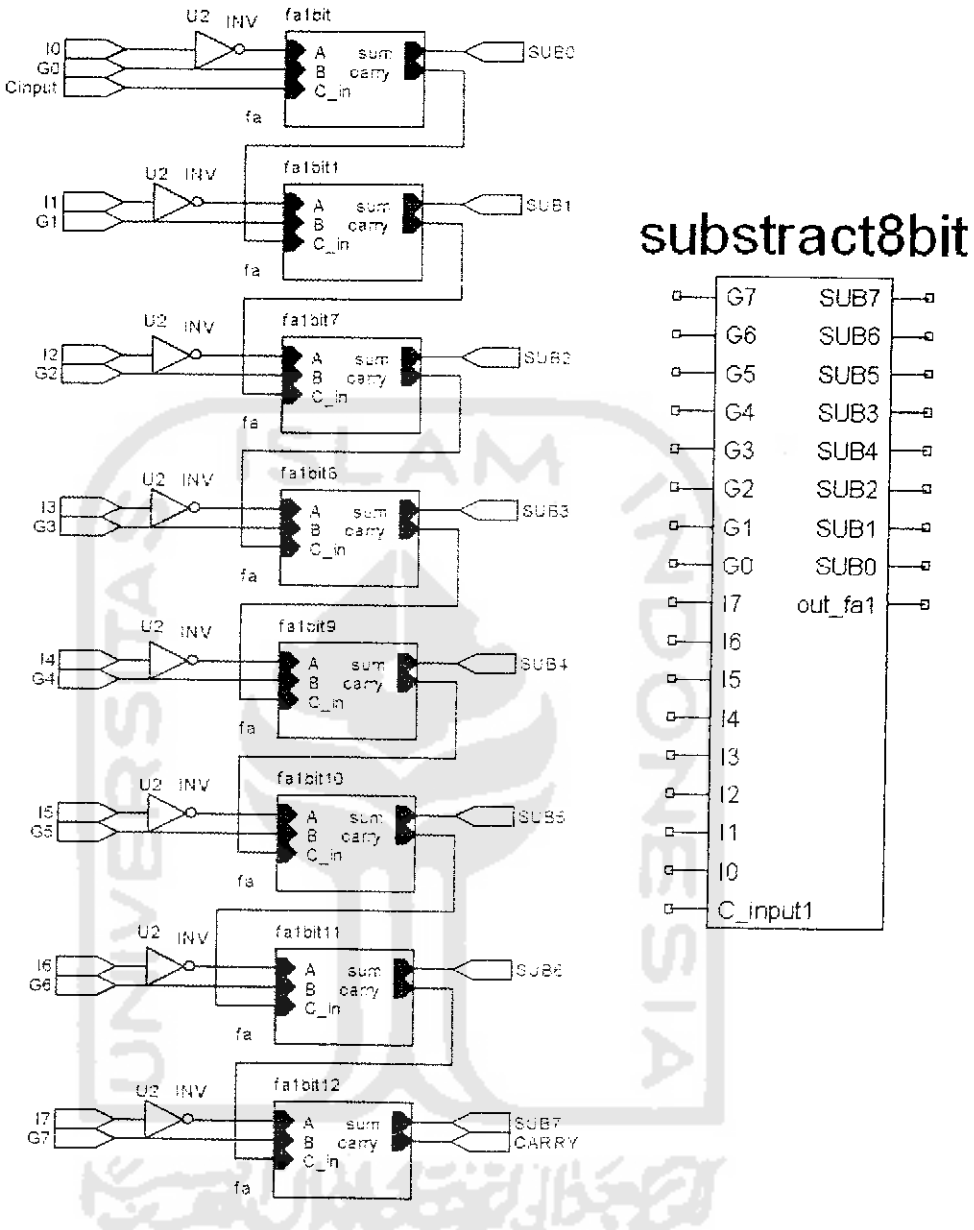
Gambar 3.8 Rangkaian dan Blok *Full Adder* 8 bit

3.3.1.3 Rangkaian Pengurang (*Substractor*)

Operasi pengurangan dengan menggunakan sistem komplemen dua sebenarnya melibatkan operasi penjumlahan. Untuk mengurangi satu bilangan biner (misalnya B) dari bilangan biner yang lain (misalnya A) dilakukan prosedur berikut :

1. Negasikan B. Hal ini akan mengubah bilangan biner B ke nilai ekivalen dengan tanda berlawanan.
2. Tambahkan hasilnya ke bilangan biner A. Hasil operasi ini akan menghasilkan selisih antara A dan B.

Rangkaian penjumlah penuh delapan bit yang telah dibuat sebelumnya pada Gambar 3.8 dapat diadopsi untuk membuat operasi pengurangan seperti yang telah dijelaskan di atas. Bentuk komplemen dua dari suatu bilangan biner dapat diperoleh dengan menginverskan masing-masing bit dan kemudian menambahkan 1 pada LSB. Gambar 3.9 berikut ini menampilkan bagaimana operasi ini diterapkan.



Gambar 3.9 Rangkaian dan Blok *Substractor* 8 bit

Pada Gambar 3.9 tersebut, masukkan I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0 diinversikan dan Cin dibuat 1. Operasi ini untuk merepresentasikan bentuk komplemen dua dari bilangan I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0. Kemudian operasi selanjutnya sama saja dengan operasi penjumlahan biasa karena rangkaian digital yang digunakan adalah

penjumlah penuh delapan bit. Namun disini masukan I bernilai negatif karena dinyatakan dengan komplemen dua. Dengan menambahkan bilangan negatif pada bilangan positif sama saja mengurangkan bilangan tersebut dari bilangan yang ditambahkan. Bawaan (*carry*) dari operasi penjumlahan terakhir tidak dipakai dalam sistem komplemen dua sehingga dapat diabaikan.

3.3.1.4 Rangkaian Pengali (*Multiplier*)

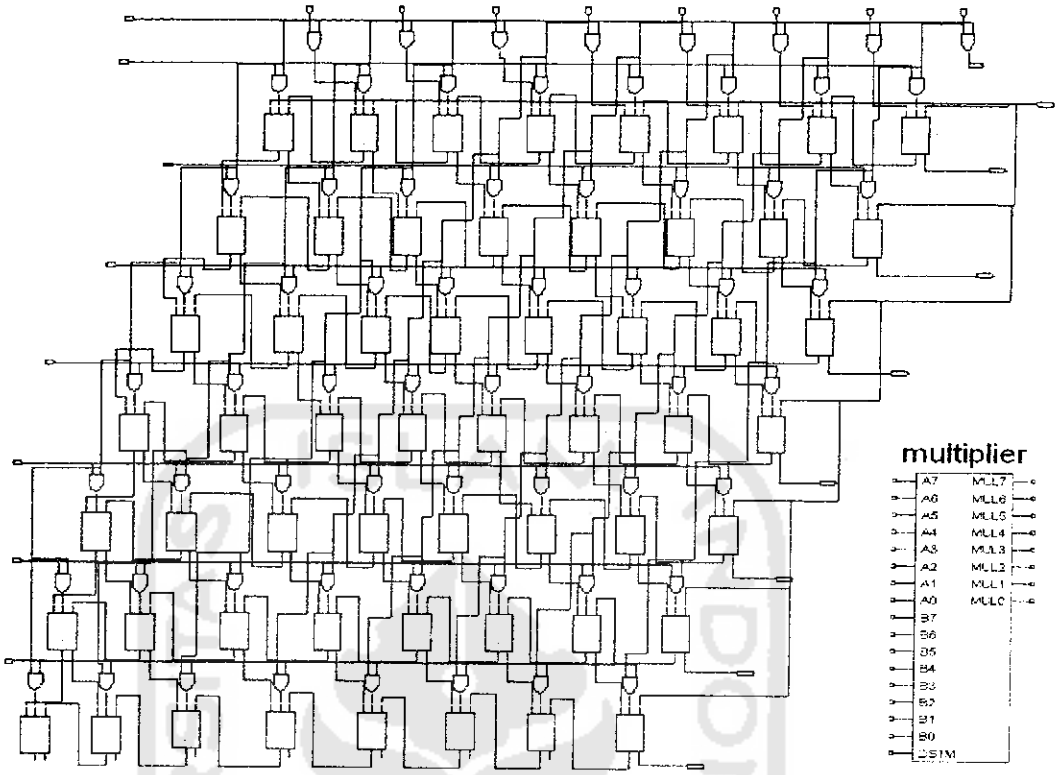
Operasi perkalian dalam bilangan biner dilakukan sama seperti dalam bilangan desimal. Proses perkalian biner ini sebenarnya lebih sederhana karena digit-digit pengalinya hanya ada 0 dan 1, sehingga hanya dengan 0 atau 1 saja dan tidak ada digit lainnya. Gambar 3.10. berikut mengilustrasikan perkalian 2 bilangan 8 bit.

$A_7A_6A_5A_4A_3A_2A_1A_0$ 00001111
$B_7B_6B_5B_4B_3B_2B_1B_0$ 00001011 x
00001111
00001111
00000000
00001111
00000000
00000000
00000000
00000000 +
000000010100101

Gambar 3.10 Perkalian Dua Bilangan Biner 8 bit

Pada gambar 3.10 operasi dumulai dengan LSB dari bilangan pengali (B_0) yaitu 1 mengalikan bilangan $A_7A_6A_5A_4A_3A_2A_1A_0$ dan menghasilkan 00001111 yang selanjutnya disebut hasil sementara ke-1. Selanjutnya bit kedua dari pengali (B_1) yaitu 1 mengalikan bilangan $A_7A_6A_5A_4A_3A_2A_1A_0$ dan menghasilkan 00001111 yang selanjutnya disebut hasil sementara ke-2. Namun hasil sementara ke-2 ini ditempatkan tergeser 1 bit ke kiri relatif terhadap hasil sementara ke-1. Bit ke tiga dari pengali (B_2) yaitu 0 mengalikan bilangan $A_7A_6A_5A_4A_3A_2A_1A_0$ dan menghasilkan 00000000 yang selanjutnya disebut hasil sementara ke-3. Hasil sementara ke-3 ini juga ditempatkan tergeser satu bit ke kiri relatif terhadap hasil sementara ke-2. Begitulah seterusnya sampai bit pengali terakhit (B_7) 0 mengalikan bilangan $A_7A_6A_5A_4A_3A_2A_1A_0$ dan menghasilkan 00000000 yang selanjutnya disebut hasil sementara sebelumnya. Pada akhirnya terdapat delapan hasil sementara dan kemudian seluruh hasil sementara ini dijumlahkan untuk memperoleh hasil akhir.

Metode inilah yang digunakan dalam perancangan perangkat keras ini. Karena dalam perancangan yang digunakan hanya delapan bit maka enam belas bit keluaran tersebut diambil hanya delapan bit LSB saja. Pada gambar 3.10 ditunjukkan rangkaian pengali yang dibuat dari rangkaian *full adder* 1 bit dan gerbang AND2.



Gambar 3.11 Rangkaian dan Blok *Multiplier* 8 bit

3.3.1.5 Rangkaian Pembagi (*Divider*)

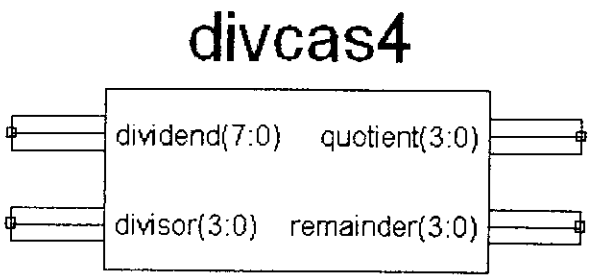
Pada prinsipnya operasi pembagian merupakan kebalikan dari operasi perkalian, yaitu bilangan *dividen*, bilangan pembagi (*divisor*) dan hasil bagi (*quotient*) secara berurutan dapat disamakan dengan hasil kali, multiplikan dan pengali. *Quotient* (hasil bagi) dikalikan dengan *divisor* (pembagi) menghasilkan *dividen* (bilangan yang dibagi).

atau $\dfrac{dividen}{divisor} = \text{hasil bagi}$

atau $divisor \;\overline{)\; dividen}^{\;\text{hasil bagi}}$

Berbeda dengan operasi perkalian, operasi pembagian tidak bersifat komutatif, artinya operan-operan tidak dapat saling ditukarkan. $A/B \neq B/A$ kecuali A = B. Disamping itu, meskipun kedua operan (*dividen* dan *divisor*) berupa bilangan bulat belum tentu menghasilkan bilangan hasil bagi berupa bilangan bulat. Hal inilah yang mendasari konsep bilangan sisa (*remainder*). Pembagian berbeda dengan perkalian dalam beberapa aspek. Pembagian merupakan operasi penggeseran-pengurangan bilangan pembagi, sedangkan pekalian merupakan operasi pergeseran-penjumlahan bilangan multiplikan. Oleh karena itu pembagian dipengaruhi oleh hasil pengurangan tahap sebelumnya dan itu terjadi secara berurutan untuk semua tahap berikutnya. Hal seperti ini tidak terjadi di dalam operasi perkalian karena untuk mendapatkan hasil, jika diperlukan dapat dilakukan secara bersamaan.

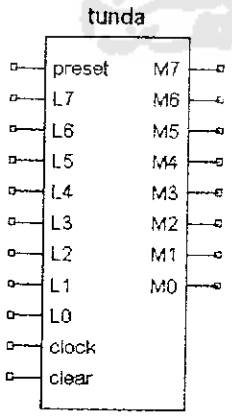
Rangkaian pembagi yang digunakan tidak dibuat sendiri melainkan mengambil sumber dari internet. Dalam operasi pembagian ini jumlah bit untuk *dividen* adalah 8 bit dan *divisor* / pembaginya 4 bit. Jadi untuk hasil (*quotient*) jumlah bitnya 4 dan untuk sisa (*remainder*) juga 4 bit. Gambar 3.12 adalah blok untuk rangkaian divider.



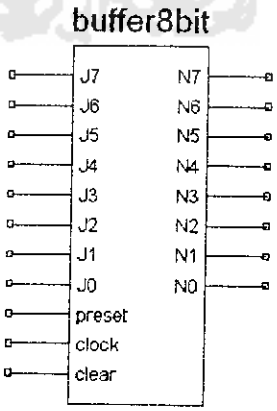
Gambar 3.12 Blok untuk Rangkaian *Divider*

3.3.1.6 Register Penyangga dan Tunda

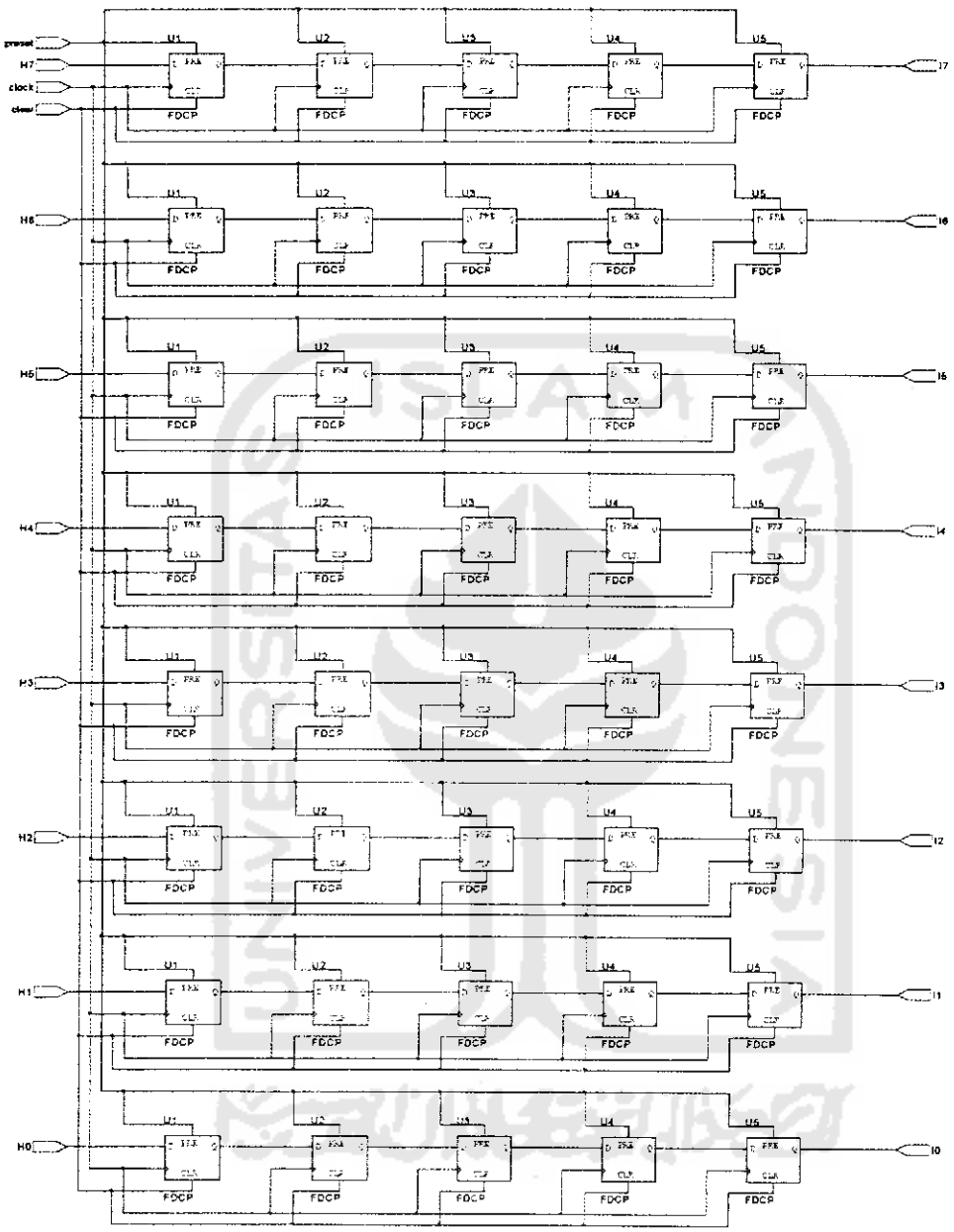
Register penyangga (*buffer*) digunakan untuk menyimpan data pada awal interval perhitungan dan menahannya sampai awal interval perhitungan berikutnya. Register penyangga diperlukan dalam perancangan perangkat keras ini untuk mensinkronkan proses aliran data sepanjang filter. Dalam implementasi filter adaptif ini diperlukan lima siklus *clock* untuk satu hasil keluaran filter. Oleh karena itu register penyangga data dalam rancangan ini digunakan untuk menyimpan data pada awal lima *clock* pertama dan menahannya sampai dihasilkan satu kali keluaran filter. Ini berarti keluaran register penyangga akan tertunda lima *clock* terhadap masukannya. Dengan demikian register penyangga ini juga dapat digunakan untuk mengimplementasikan tunda. Register penyangga dapat dibuat dari rangkaian FLIP FLOP D. Untuk membuat register penyangga lima *clock* dan dengan kapasitas data delapan bit maka dibutuhkan empat puluh buah FLIP FLOP D. Gambar 3.13 dan 3.14 adalah blok digital untuk rangkaian tunda dan penyangga. Sedangkan untuk gambar 3.15 adalah isi dari gambar 3.13 dan gambar 3.14.



Gambar 3.13 Blok Tunda



Gambar 3.14 Blok untuk Penyangga



Gambar 3.15 Rangkaian Penyangga dan Tunda

Pada gambar 3.15 (gambar rangkaian penyangga) terlihat bahwa delapan buah FLIP FLOP D pararel digunakan masing-masing untuk satu bit data. Keluaran Q sel pertama dimasukkan ke FLIP FLOP D pararel sel kedua. Keluaran Q sel kedua dimasukkan ke FLIP FLOP D pararel sel ketiga. Begitu seterusnya

sampai FLIP FLOP D pararel yang kelima. Dengan demikian maka keluaran pararel Q dari sel terakhir akan tertunda lima *clock* masukan awal D di sel pertama.

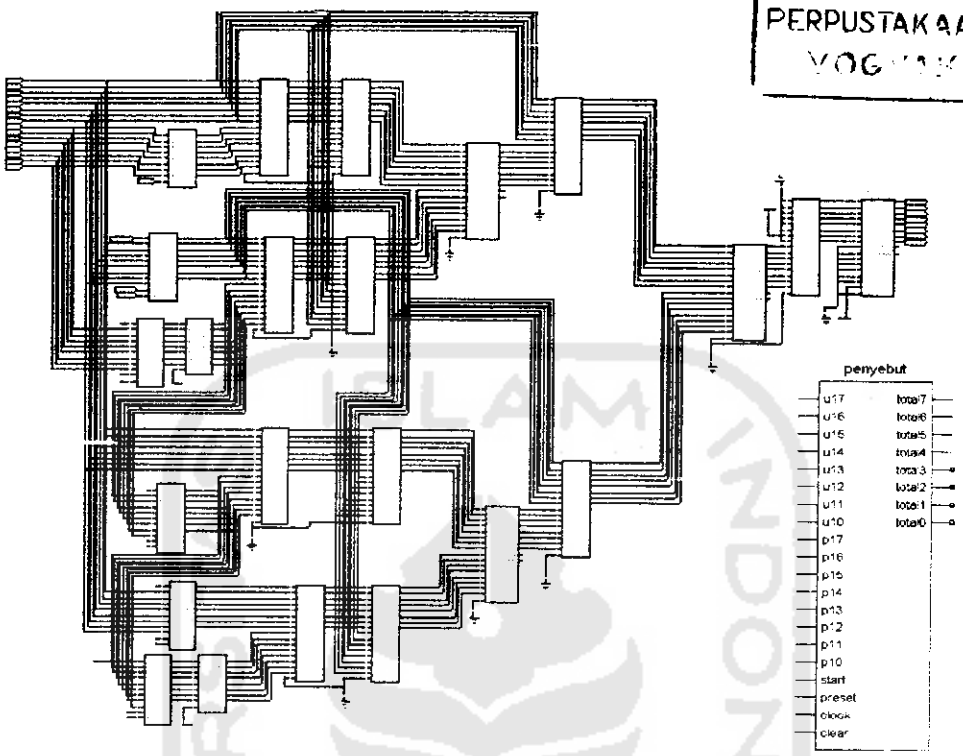
3.3.2 Perancangan Filter dengan Algoritma RLS

3.3.2.1 Rangkaian untuk *Gain Vector* (k)

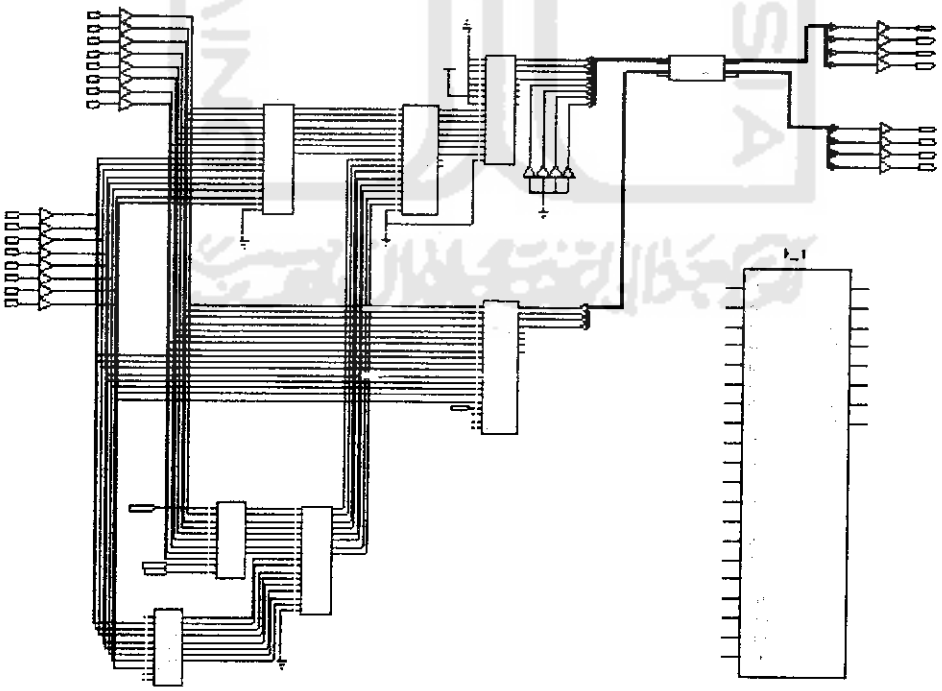
Berdasarkan persamaan 2.14 *Gain vector* ini mempunyai rumus

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

Untuk memudahkan perancangan, maka penyebut dari rumus diatas dipisahkan menjadi blok sendiri yang diberi nama blok penyebut. Gambar 3.16 adalah rangkaian penyebut untuk λ = 0,5. Untuk perancangan λ yang lain tinggal mengubah data input pada rangkaian multiplier terakhir. Untuk rangkaian ini mempunyai input u[7:0], p[7:0], *start*, *preset*, *clock* dan *clear*. Sedangkan hasil dari operasi ini akan diambil 4 bit MSB untuk digunakan sebagai *divisor*. Sedang pembilang dari rumus k diatas merupakan *dividen* untuk operasi pembagian. Namun untuk operasi ini, pembilang yang digunakan adalah MSB yang dijadikan LSB kemudian MSBnya diubah menjadi nol. Untuk rangkaian secara keseluruhan dari rangkaian *gain vector* dapat dilihat pada gambar 3.17



Gambar 3.16 Rangkaian dan Blok Penyebut



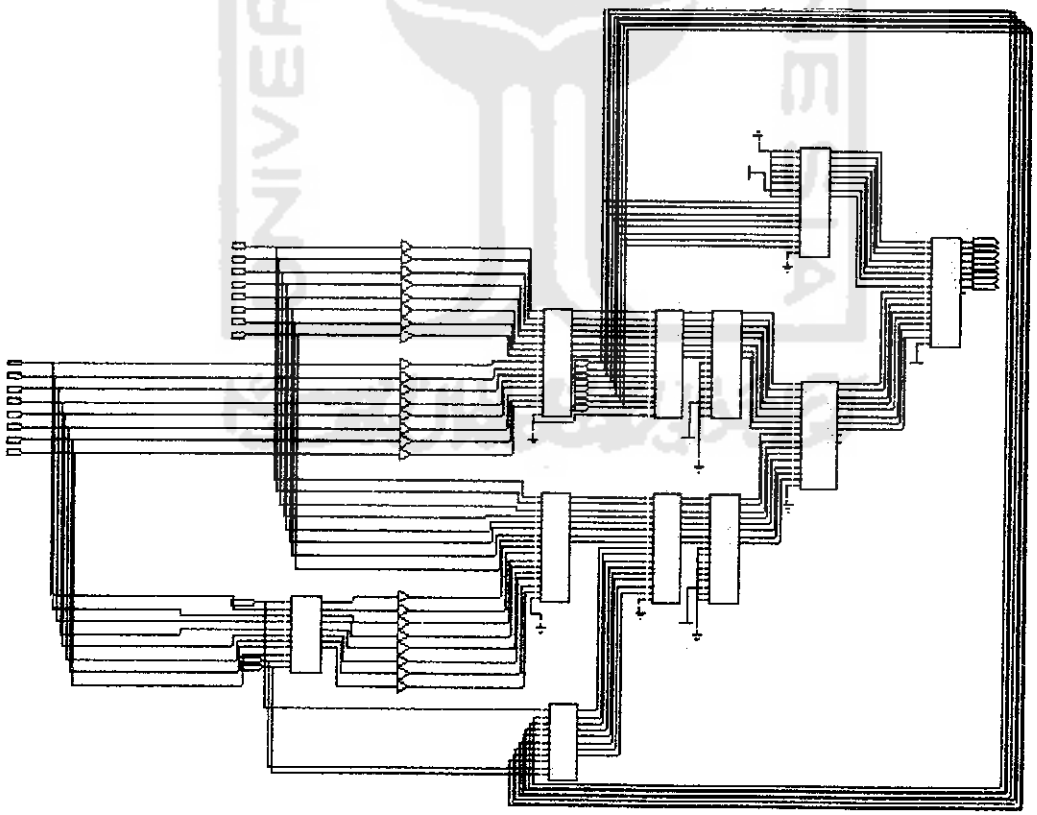
Gambar 3.17 Rangkaian dan Blok *Gain Vector* (k)

3.3.2.2 Rangkaian untuk Matriks Korelasi (P)

Berdasarkan persamaan 2.17 matriks korelasi mempunyai rumus

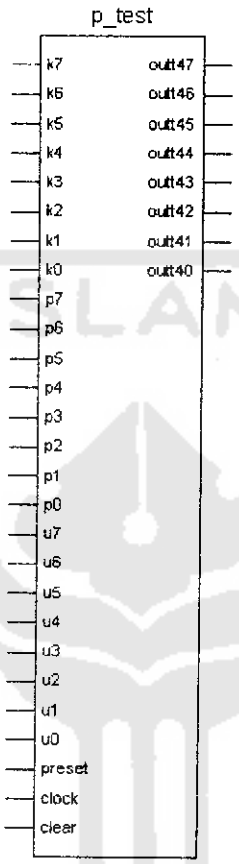
$$P(n) = \lambda^{-1}P(n-1) - \lambda^{-1}k(n)u^{H}(n)P(n-1)$$

Untuk membuat rangkaian digital dari rumus diatas diperlukan 7 blok *multiplier*, 1 blok *full adder* 8 bit, 1 blok *substractor*, 2 blok tunda dan 24 *buffer* 1 bit. Rangkaian inverse corelation matrix membutuhkan input k[7:0], p[7:0], u[7:0], *preset*, *clock* dan *clear*. Sedangkan untuk outputnya adalah out4[7:0] yang nantinya akan bercabang 2 yaitu masuk ke input blok k dan satunya akan diperbaharui lagi untuk dijadikan input untuk blok p itu sendiri. Gambar 3.18 menunjukkan rangkaian digital untuk matriks korelasi (P).



Gambar 3.18 Rangkaian Matriks Korelasi (P)

Sedangkan untuk blok dari rangkaian matriks korelasi ditunjukkan pada gambar 3.19



Gambar 3.19 Blok dari Rangkaian Matriks korelasi (P)

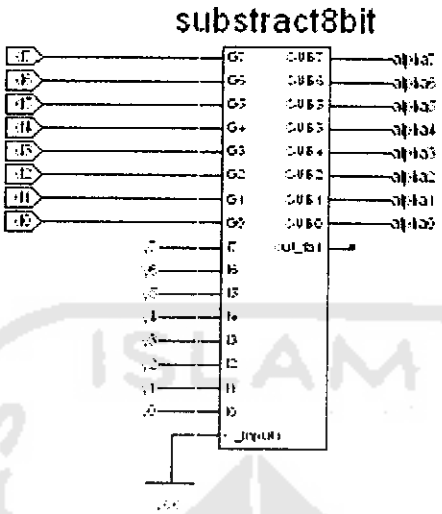
3.3.2.3 Rangkaian untuk *Error* atau Alpha (α)

Berdasarkan persamaan 2.15 alpha mempunyai rumus

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

$$= d(n) - y(n)$$

Untuk membuat rangkaian digital dari rumus diatas hanya diperlukan 1 blok *substractor*. Input dari rangkaian ini adalah d[7:0] dan keluaran filter yang diumpan balik sebagai pengurang. Gambar 3.20 adalah rangkaian untuk alpha (α).



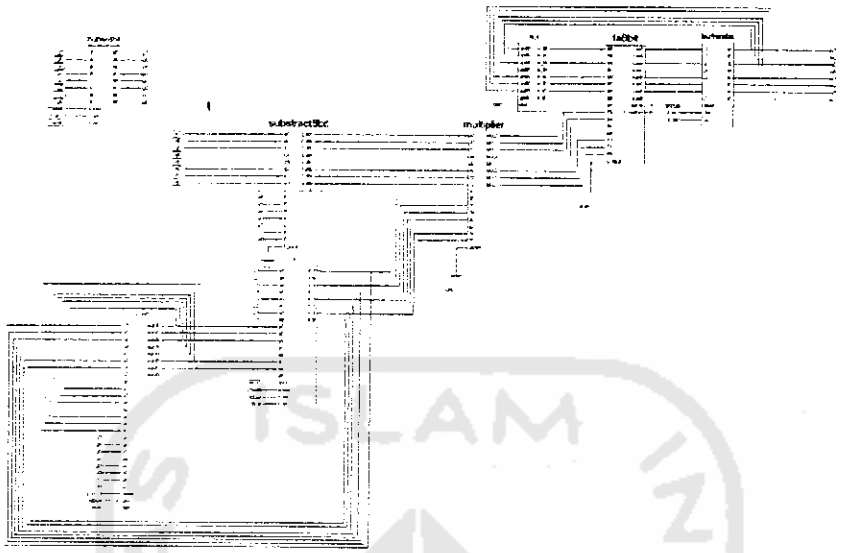
Gambar 3.20 Rangkaian Alpha (α)

3.3.2.4 Rangkaian untuk Pembaharuan Bobot (w)

Berdasarkan persamaan 2.13 pembaharuan bobot mempunyai rumus

$$w(n) = w(n-1) + k(n)\alpha(n)$$

Untuk membuat rangkaian digital dari rumus diatas diperlukan 1 blok p, 1 blok k, 1 blok untuk nilai inisial = 0, 1 blok *substractor*, 1 blok *full adder*, 2 blok *buffer* dan 1 blok *multiplier*. Rangkaian ini membutuhkan input d[7:0] dan u[7:0]. Sedangkan untuk outputnya akan diperbaharui dan digunakan sebagai input lagi. Gambar 3.21 menunjukkan rangkaian untuk 1 bobot.



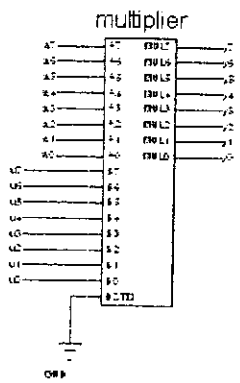
Gambar 3.21 Rangkaian untuk 1 Bobot

3.3.2.5 Rangkaian untuk Output Filter (y)

Berdasarkan persamaan 2.3 output filter mempunyai rumus

$$y_n = w_n . u_n$$

Untuk membuat rangkaian digital dari rumus diatas hanya diperlukan 1 blok *multiplier*, yang membutuhkan input w[7:0] dan u[7:0]. Jika perancangan filter ini menggunakan 2 bobot maka untuk menghitung output filter secara keseluruhan, harus dijumlahkan antara output filter 1 dan output filter 2. Gambar 3.22 menunjukkan gambar rangkaian 1 output filter.



Gambar 3.22 Rangkaian 1 Output Filter