

TUGAS BESAR

MATA KULIAH

EC 7030 Metode Formal

Dosen : Ir. Budi Rahardjo, MSc, PhD

PEMODELAN DAN SIMULASI

SHIFT REGISTER DENGAN PROMELA DAN SPIN



Oleh

Ivan Suci Firmansyah

23202010

PROGRAM PASCA SARJANA

DEPARTEMEN TEKNIK ELEKTRO

INSTITUT TEKNOLOGI BANDUNG

2005

Daftar Isi

Daftar Isi.....	2
1 Pendahuluan.....	3
1.1 Spesifikasi Masalah.....	3
1.2 Batasan Masalah.....	3
2 Pemodelan Sistem.....	3
2.1 Flip-flop sebagai pembentuk register.....	3
2.2 Register Geser (Shift register).....	8
2.3 Model Kerja Shift Register.....	11
3 Simulasi dan Verifikasi.....	12
3.1 Pemodelan menggunakan Promela dan SPIN.....	12
3.2 Hasil Simulasi.....	18
4 Kesimpulan.....	20
Daftar Pustaka.....	22
Lampiran : Hasil Simulasi.....	23

1 Pendahuluan

1.1 Spesifikasi Masalah

Perangkat keras memori secara logik merupakan implementasi dari rangkaian logika register yang terdiri dari rangkaian flip-flop, baik yang pengiriman datanya dilakukan secara paralel maupun serial. Dalam tugas ini dilakukan pemodelan cara kerja untuk memori dengan menggunakan shift register dengan D flip-flop.

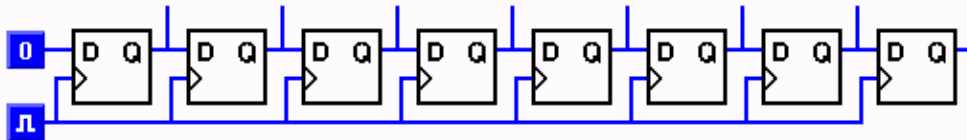


Diagram diatas memperlihatkan D Flip-flop yang tersambung dalam sebuah rangkaian SERIAL IN, SERIAL OUT shift register. Setiap datangnya pulsa clock, data dari input D dari masing-masing flip-flop akan di transfer kepada Q output. Jika 1 merupakan input dari flip-flop yang pertama, maka pada pulsa berikutnya 1 akan ditransfer ke output flip-flop 1 dan sekaligus menjadi input bagi flip-flop 2. begitu seterusnya.

1.2 Batasan Masalah

Pada pemodelan ini digunakan 4 buah D flip-flop yang akan mengalami satu buah siklus pulsa clock sehingga hanya terjadi 1 kali pergeseran data. Shift register menggunakan satu buah Temp register berbentuk D flip-flop juga yang digunakan sebagai penerima keluaran dari shift register.

2 Pemodelan Sistem

2.1 Flip-flop sebagai pembentuk register

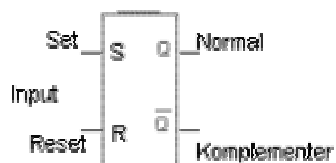
Rangkaian logika dikelompokkan dalam dua kelompok besar. Kelompok-kelompok gerbang logika dikenal sebagai rangkaian logika kombinasional. Kelompok lain dikenal sebagai rangkaian logika sekuensial yang berdasarkan para rangkaian flip-flop yang

sangat bermanfaat karena memiliki karakteristik memori. Flip-flop dapat dirangkai dari gerbang logika, juga dapat diperoleh dalam bentuk IC. Flip-flop diinterkoneksi untuk membentuk rangkaian logika sekuensial untuk penyimpanan, pewaktu, penghitungan dan pengurutan (sequencing). Terdapat beberapa jenis flip-flop dan kita akan mempelajari beberapa diantaranya.

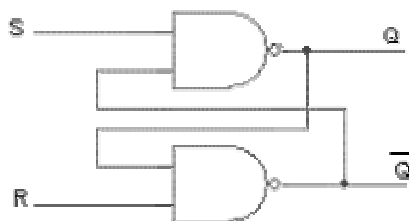
A. FLIP-FLOP RS

Kebanyakan flip-flop dasar merupakan flip-flop RS. Simbol logika untuk flip-flop RS ditunjukkan pada Gambar 1. Simbol logika tersebut memiliki dua input yang diberi label S (set) dan R (reset) di sebelah kiri. Flip-flop RS pada simbol ini memiliki input aktif low/nol yang ditunjukkan dengan adanya gelembung-gelembung kecil pada input R dan S. Tidak seperti gerbang logika, flip-flop memiliki dua output komplementer. Output tersebut diberi label Q dan \bar{Q} . Output Q dianggap merupakan output normal, dan dalam kondisi normal kedua output selalu merupakan komplementer. Karena fungsi flip-flop memegang data sementara, maka flip-flop ini sering disebut RS Latch Flip-Flop.

Flip-flop RS dapat disusun dari gerbang gerbang logika. Penyusunan flip-flop RS dari dua gerbang NAND dapat dilihat pada Gambar 2.



Gambar 1 Simbol Logika flip-flop RS



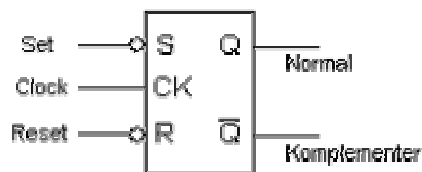
Gambar 2 Flip-flop dari gerbang NAND

B. FLIP-FLOP DETAK RS (CLOCKED RS FLIP-FLOP)

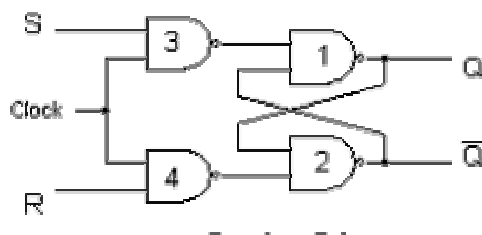
Latch flip-flop pada dasarnya merupakan suatu piranti asinkron (asynchronous). Peralatan seperti itu tidak beroperasi serempak dengan detak atau piranti pewaktu. Bila input diaktifkan, maka output normal segera diaktifkan seperti pada rangkaian logika kombinasional.

Flip-flop detak RS menambahkan suatu sifat sinkron yang berguna untuk latch RS. Flip-flop detak RS akan beroperasi serempak dengan detak atau piranti pewaktu. Dengan kata lain, flip-flop tersebut beroperasi secara sinkron. Simbol logika untuk flip-flop detak RS diilustrasikan pada Gambar 3. Flip-flop ini memiliki tambahan input detak CK (Clock).

Flip-flop detak RS juga dapat dibangun dari gerbang NAND seperti pada Gambar 4. Gerbang NAND 3 dan 4 menambahkan sifat berdetak pada latch RS tersebut. Input detak / clock akan memacu (membuka) flip-flop bila pulsa detak menjadi high. Setiap kali pulsa detak menjadi high, maka informasi pada input data (R dan S) akan dipindahkan ke output.



Gambar 3 Simbol Logika flip-flop Detak RS



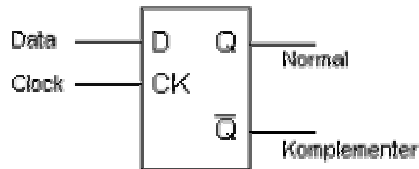
Gambar 4 Flip-flop Detak RS dari gerbang NAND

C. FLIP-FLOP D

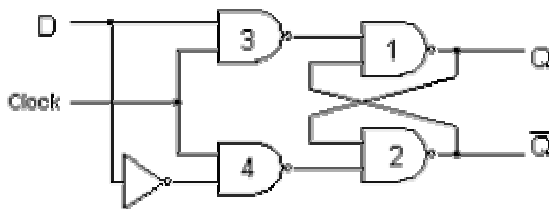
Flip-flop D (lihat Gambar 5) hanya memiliki input data tunggal (D) dan input detak (CK). Flip-flop D sering kali disebut juga sebagai flip-flop tunda. Nama ini menggambarkan operasi unit ini. Apapun bentuk input pada input data (D), input tersebut akan tertunda

selama satu pulsa detak untuk mencapai output normal (Q). Data dipindahkan ke output pada transisi detak Low ke High.

Flip-flop detak RS dapat juga diubah menjadi flip-flop D dengan menambahkan suatu inverter. Bila dibangun dari gerbang NAND, maka rangkaianannya menjadi seperti gambar 6.



Gambar 5 Simbol Logika Flip-flop D



Gambar 6 Flip-flop D dari gerbang NAND

D. FLIP - FLOP JK

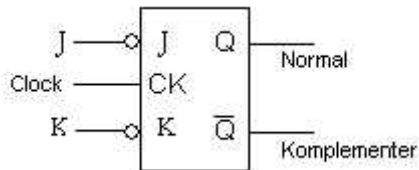
Flip-flop ini dapat dianggap sebagai flip-flop universal, karena flip-flop jenis lain dapat dibuat dari flip-flop JK. Simbol logika pada Gambar 7 mengilustrasikan tiga input sinkron (J, K dan CK). Input J dan K merupakan input data, dan input clock memindahkan data dari input ke output. Diperlukan keseluruhan pulsa (bukan sekedar tansisi low ke high atau high ke low saja) untuk memindahkan data dari input ke output.

Dua sifat unik dari flip-flop JK adalah:

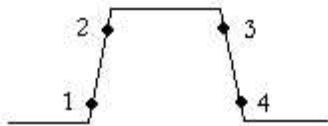
1. Jika kedua data input pada keadaan nol, tidak akan terjadi perubahan pada output meskipun diberikan sinyal clock (output tetap).
2. Jika kedua data input pada keadaan satu, pada tiap pulsa clock data output akan berubah dari sebelumnya (komplemen dari data sebelumnya).

Kita dapat membangun suatu flip-flop JK dari gerbang NAND seperti Gambar 9. Nampak bahwa sebenarnya flip-flop JK terdiri dari dua flip-flop yang terangkai menjadi satu. Flip-flop yang kedua (slave-budak) mengikuti keadaan yang ditentukan oleh flip-flop yang pertama (master-tuan). Suatu transisi hanya dapat terjadi dengan satu pulsa clock penuh. Kejadian dibawah ini terjadi selama urutan pemacuan pulsa pada titik-titik bernomor pada Gambar 8.

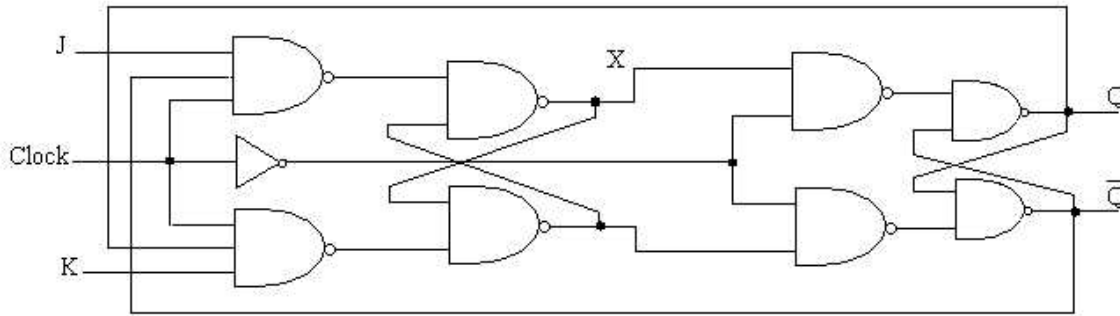
1. Master dan Slave diisolasi/dipisahkan.
2. Data dimasukkan dari input J dan K, berhenti di Master.
3. Input J dan K Master ditutup.
4. Slave mengambil data dari output Master.



Gambar 7 Simbol Logika Flip-flop Detak JK



Gambar 8 Pulsa-pulsa Clock



Gambar 5.9
Flip-flop JK dari gerbang NAND

Gambar 9 Flip-flop JK dari gerbang NAND

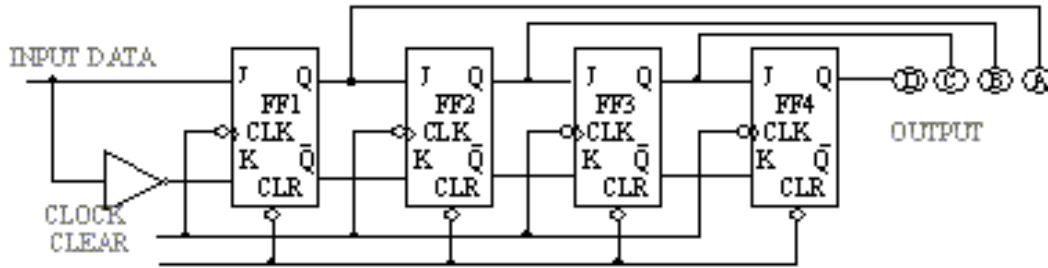
2.2 Register Geser (Shift register)

Register geser (shift register) merupakan salah satu piranti fungsional yang banyak digunakan dalam sistem digital. Tampilan pada layar kalkulator dimana angka bergeser ke kiri setiap kali ada angka baru yang diinputkan menggambarkan karakteristik register geser tersebut. Register geser ini terbangun dari flip-flop. Register geser dapat digunakan sebagai memori sementara, dan data yang tersimpan didalamnya dapat digeser ke kiri atau ke kanan. Register geser juga dapat digunakan untuk mengubah data seri ke paralel atau data paralel ke seri.

A. Register Geser Beban Seri

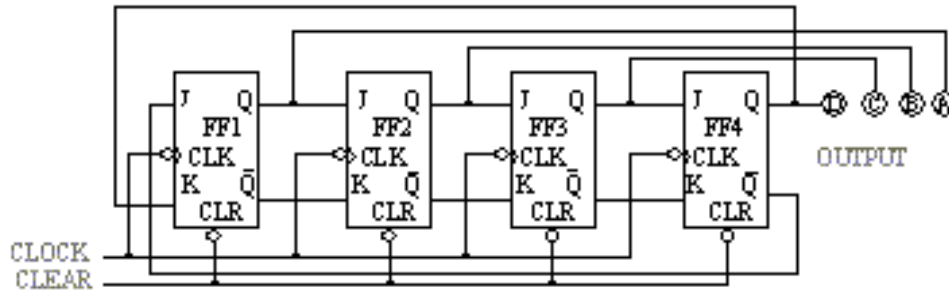
Suatu register geser 4 bit sederhana diilustrasikan pada Gambar 10. Perhatikan penggunaan empat flip-flop JK sebagai flip-flop D pada rangkaian tersebut. Bit data (0 dan 1) dimasukkan ke dalam input J dari FF1. Input reset/clear akan mereset semua flip-flop ke logika 0 bila di aktifkan dengan level Low. Pulsa pada input clock akan menggeser data dari input data seri ke posisi A(Q dari FF1). Indikator (A, B, C, D) menunjukkan isi masing-masing flip-flop.

Kalau diasumsikan semua flip-flop semuanya direset ($Q=0$), maka output akan menjadi 0000. Beri logika 1 pada input prereset dan pada input data. Kita berikan satu pulsa pada input clock. Maka output akan menunjukkan 1000 ($A=1, B=0, C=0, D=0$). Kita masukkan sekarang logika 0 pada input data. Setelah diberi pulsa clock lagi, output akan menunjukkan 0100. Hal ini menunjukkan terjadinya penggeseran data secara serial. Begitu seterusnya.

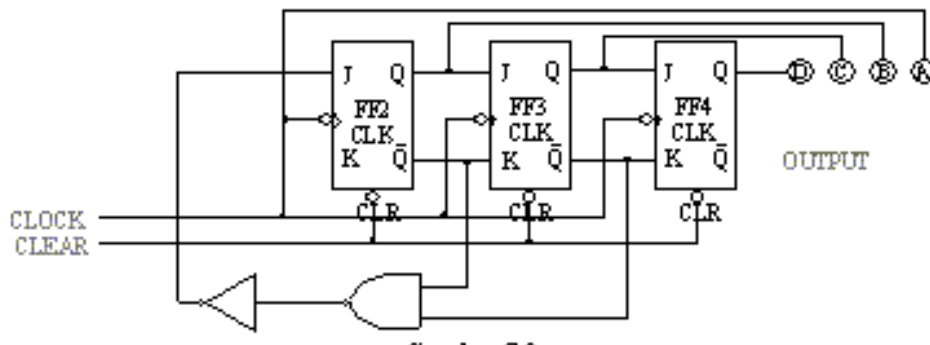


Gambar 10 Diagram Logika Shift Register Beban Seri 4 bit

Untuk model diagram logika diatas, kita harus memasukkan data pulsa ke J FF1, yang akan digeser hingga FF4. Bila diinginkan suatu data yang terus berputar, dipakailah Ring Counter. Pada prinsipnya sama dengan register geser biasa, hanya outputnya diumpanbalikkan ke input sehingga terjadi siklus yang terus menerus. Sebagai contoh adalah rangkaian Switch Tail Ring Counter dan Register Geser Resirkulasi 3-bit yang diilustrasikan pada Gambar 11 dan 12.



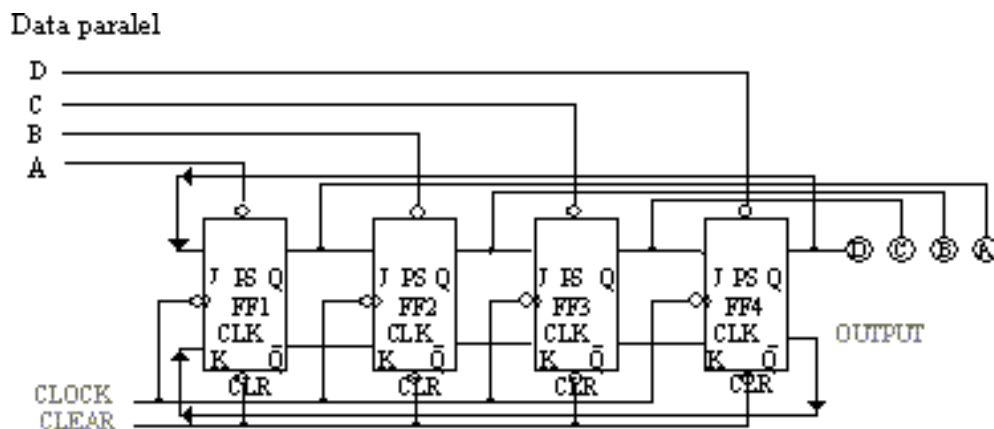
Gambar 11 Diagram Logika Switch Tail Ring Counter



Gambar 12 Diagram Logika Register Geser Resirkulasi 3 bit

B. Register Geser Beban Paralel

Kelemahan register geser seri adalah bahwa untuk membebani register tersebut diperlukan banyak pulsa clock. Suatu register geser beban paralel membebani semua bit informasi dengan segera. Salah satu register geser beban paralel yang sederhana ditunjukkan pada Gambar 13. Perhatikan adanya umpan balik yang melintas dari output FF4 kembali masuk ke input FF1. Garis ini merupakan garis perputaran kembali, dan lintasan tersebut akan menyimpan data yang secara normal akan hilang keluar ke ujung kanan dari register tersebut. Dengan kata lain data akan berputar kembali melalui register tersebut.



Gambar 13 Diagram Logika Register Geser Kanan yang Bersirkulasi Beban Paralel

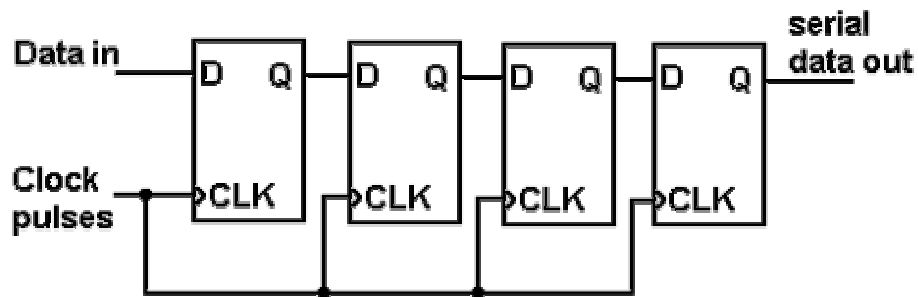
C. Register Geser Universal

Register geser tipe ini merupakan suatu register geser 4-bit yang memiliki input serial dan paralel, output paralel, mode kontrol (shift left-geser kiri dan shift right-geser kanan) serta 2 input clock. Register ini dapat bekerja pada beberapa mode kerja tergantung pengaturan mode kontrol dan input serial atau paralel yang diberikan.

Jika akan memilih geser kanan atau kiri tinggal mengatur lewat mode kontrol, dimana logika 1 berarti geser kiri, sedang logika 0 menyiapkan register untuk bekerja pada mode geser kanan.

2.3 Model Kerja Shift Register

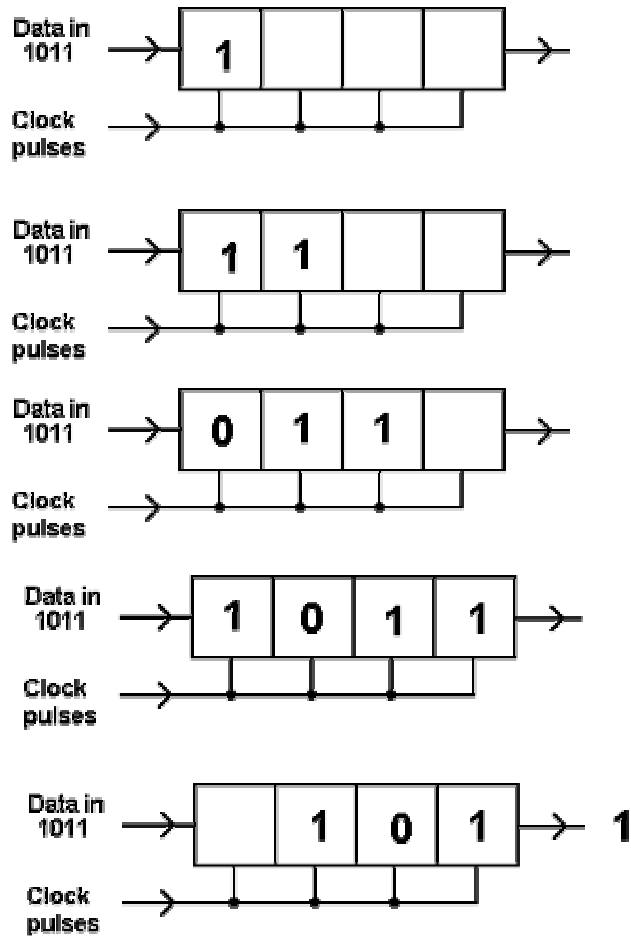
Diagram dibawah ini memperlihatkan 4 D-Flip-flop yang tersambung dalam sebuah rangkaian SERIAL IN, SERIAL OUT shift register.



Setiap datangnya pulsa clock, data dari input D dari masing-masing flip-flop akan di transfer kepada Q output. Pada awalnya, isi dari register diset 0 dengan mengirimkan clock pada CLEAR.

Jika 1 merupakan input dari flip-flop yang pertama, maka pada pulsa berikutnya 1 akan ditransfer ke output flip-flop 1 dan sekaligus menjadi input bagi flip-flop 2.

Setelah 4 pulsa clock, 1 telah menjadi output dari flip-flop 4. sehingga 4 bit data telah tersimpan di register. Pada 4 pulsa clock berikutnya semua data telah keluar dari register. Dibawah ini merupakan ilustrasi bagaimana data disimpan dalam register dengan menggunakan shift register.



Gambar 14 Simulasi Pemasukan Data Secara Serial pada Register

3 Simulasi dan Verifikasi

3.1 Pemodelan menggunakan Promela dan SPIN

Pada pemodelan yang disimulasikan ini shift register yang digunakan adalah D Flip-flop. Digunakan sebuah register Temporary bertipe D-flipflop yang juga digunakan pada sebagai umpan balik input bagi D flip-flop pertama. Tipe message dan variable dalam spin adalah sebagai berikut :

```
#define N 5 /* 4 D Latch + 1 buffer shift */
#define I 3 /* D given the smallest number */
```

Pemodelan Shift Register

```
#define L 10 /* size of buffer (>= 2*N) */

mtype = { goshift, readyshift, shifting };
chan q[N] = [L] of { mtype, byte};
```

sedangkan untuk listing program dari dari D flip-flop yang juga digunakan sebagai Temp Register adalah sebagai berikut :

```
proctype D (chan in, out; byte registernumber)
{
    bit Active = 1, know_shifting = 0;
    byte nr, maximum = registernumber, neighbourR;

    xr in;
    xs out;

    printf("MSC: %d\n", registernumber);
    out!goshift(registernumber);
end: do
    :: in?goshift(nr) ->
        if
            :: Active ->
                if
                    :: nr != maximum ->
                        out!readyshift(nr);
                        neighbourR = nr
                    :: else ->
                        /* D Register merupakan register terbesar
                        sehingga perlu shift data register 0 */
                        assert(nr == N);
                        know_shifting = 1;
                        out!shifting,nr;
                fi
            :: else ->
                out!goshift(nr)
        fi
```

```
:: in?readysift(nr) ->
    if
        :: Active ->
            if
                :: neighbourR > nr && neighbourR > maximum ->
                    maximum = neighbourR;
                    out!goshift(neighbourR)
                :: else ->
                    Active = 0
            fi
        :: else ->
            out!readysift(nr)
    fi
:: in?shifting,nr ->
    if
        :: nr != registernumber ->
            printf("MSC: SHIFT\n");
        :: else ->
            printf("MSC: SHIFT LOOP\n");
            nr_leaders++;
            assert(nr_leaders == 1)
    fi;
    if
        :: know_shifting
        :: else -> out!shifting,nr
    fi;
    break
od
}

proctype T (chan in, out; byte registernumber)
{
    bit Active = 1, know_shifting = 0;
    byte nr, maximum = registernumber, neighbourR;
```

```
    xr in;
    xs out;

    printf("MSC: %d\n", registernumber);
    out!goshift(registernumber);
end: do
    :: in?goshift(nr) ->
        if
            :: Active ->
                if
                    :: nr != maximum ->
                        out!readyshift(nr);
                        neighbourR = nr
                    :: else ->
                        /* D Register merupakan register terbesar
                        sehingga perlu shift data register 0 */
                        assert(nr == N);
                        know_shifting = 1;
                        out!shifting,nr;
                    fi
                :: else ->
                    out!goshift(nr)
                fi
        :: in?readyshift(nr) ->
            if
                :: Active ->
                    if
                        :: neighbourR > nr && neighbourR > maximum ->
                            maximum = neighbourR;
                            out!goshift(neighbourR)
                        :: else ->
                            Active = 0
```

```
        fi
        :: else ->
            out!readyshift(nr)
        fi
    :: in?shifting,nr ->
        if
        :: nr != registernumber ->
            printf("MSC: SHIFT\n");
        :: else ->
            printf("MSC: SHIFT LOOP\n");
            nr_leaders++;
            assert(nr_leaders == 1)
        fi;
        if
        :: know_shifting
        :: else -> out!shifting,nr
        fi;
        break
    od
}
```

Pada saat sebuah D flip flop dengan status aktif yang telah menerima pulsa dari generator maka D input akan mengirimkan sinyal **goshifting** pada Q output (dalam simulasi sama dengan D input flipflop selanjutnya). Setelah semua menerima acknowledgment (sinyal readyshifting) dari Q output maka secara paralel melakukan **shifting** data yang ada pada registernya. Inisiasi dari program ini adalah sebagai berikut :

```
init {

    atomic {
```

Pemodelan Shift Register

```
run D(q[0],q[1],1);
run D(q[1],q[2],2);
run D(q[2],q[3],3);
run D(q[3],q[4],4);

run Temp(q[4],q[0],5);

}
}
```

Verifikasi yang dilakukan menggunakan asumsi bahwa setelah D flip-flop yang terakhir telah menerima pulsa maka dibangkitkan signal untuk mengirimkan secara bersamaan data yang ada pada flip-flop :

```
if
  :: Active ->
    if
      :: nr != maximum ->
        out!readyshift(nr);
        neighbourR = nr
      :: else ->
        /* D Register merupakan register terbesar
           sehingga perlu shift data register 0 */
        assert(nr == N);
        know_shifting = 1;
        out!shifting,nr;
    fi
  :: else ->
    out!goshift(nr)
fi
```

dan untuk mengetahui bahwa telah terjadi perpindahan data maka dilakukan assert bahwa flip-flop 1 akan terisi oleh Temp Flip-flop :

```
if
```

```
    :: nr != registernumber ->
        printf("MSC: SHIFT\n");
    :: else ->
        printf("MSC: SHIFT LOOP\n");
        nr_leaders++;
        assert(nr_leaders == 1)
fi;
```

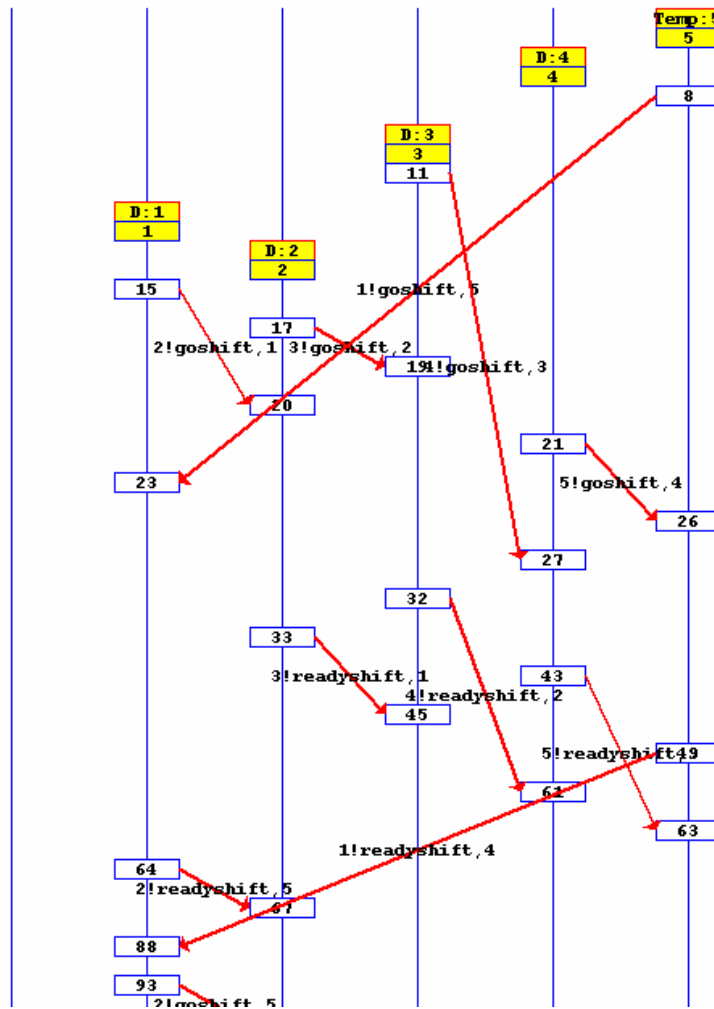
3.2 Hasil Simulasi

Hasil Simulasi terlampir.

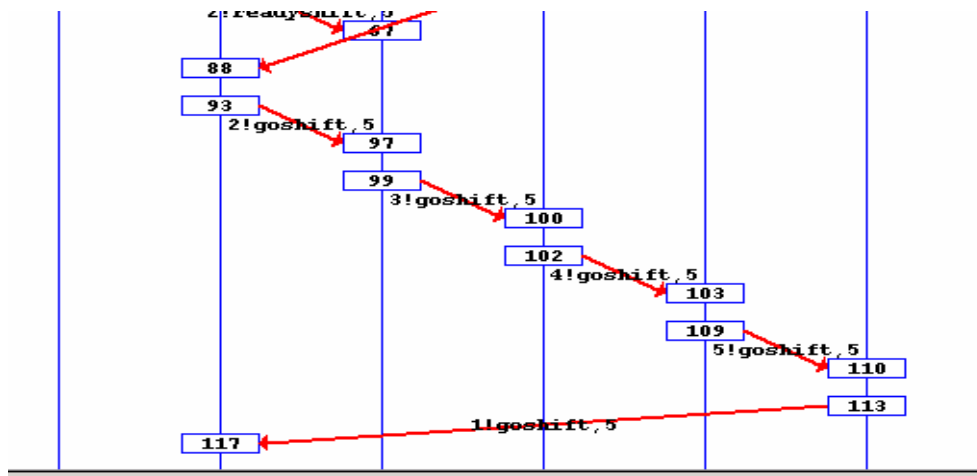
Pada Message Sequence Chart terdapat tiga fase message passing yang dilakukan :

1. Pengiriman signal & acknowledgement pengiriman data pada flip-flop tetangga

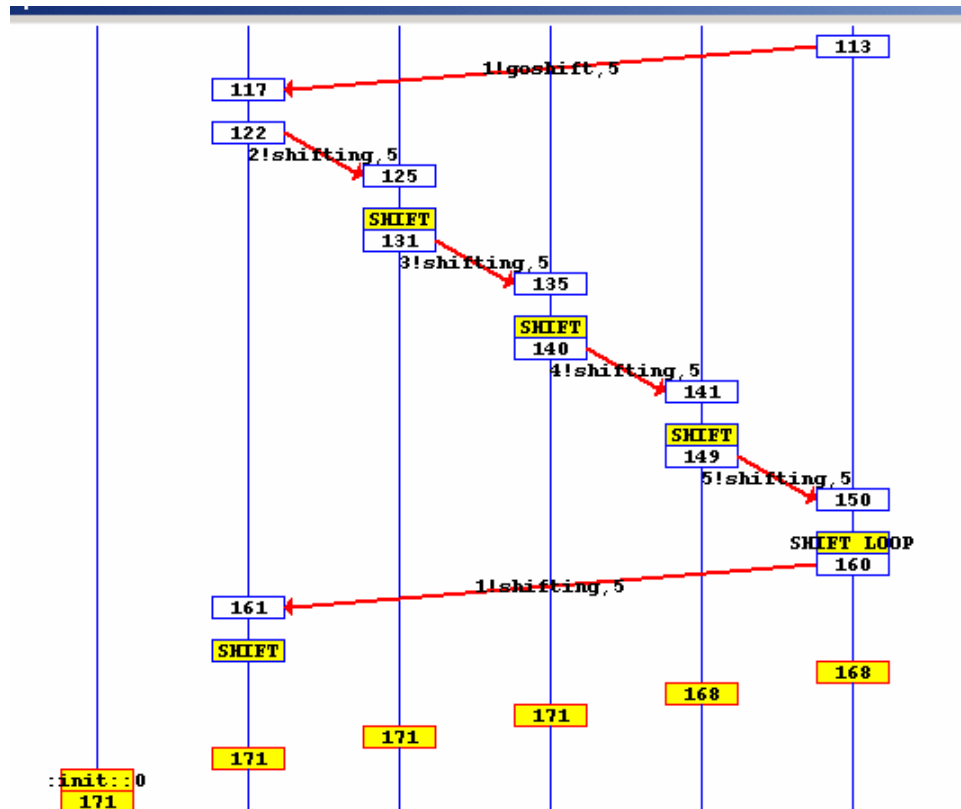
Sequence Chart



2. Pengiriman signal pengiriman data



3. shifting data



Pada diagram diatas dapat disimpulkan bahwa data terkirim ke flip-flop tetangga setelah semua flip-flop siap menerima data (clock pulsa terjadi).

4 Kesimpulan

Berdasarkan simulasi diatas PROMELA dan spin bisa digunakan untuk melakukan verifikasi atas logik hardware dalam hal ini shift register yang memiliki fungsi sebagai memori. Pendekatan pemodelan ini dapat gunakan secara lebih luas untuk rangkaian digital sehingga verifikasi dari spesifikasi rangkaian logika hardware bisa dilakukan.

Daftar Pustaka

1. Shift Register, Informasi diperoleh secara online di http://www.play-hookey.com/digital/shift-out_register.html, Ken Bigelow, 2000-2004
2. Diktat Petunjuk Praktikum Elektronika Digital , Jurusan Teknik Elektro FT UWM Surabaya
3. http://ourworld.compuserve.com/homepages/g_knott/elect339.htm , Graham Knott 1999
4. Spin Verifier's Roadmap : BUILDING AND VERIFYING Spin MODELS, Informasi dapat diperoleh secara online di <http://spinroot.com/spin/Man/roadmap.html>
5. Budi Raharjo, Paper Metoda Formal, Institut Teknologi Bandung, 2002
6. Gerard J. Holzmann, Design and Validation of Computer Protocol, Prentice Hall,1991

Lampiran : Hasil Simulasi

Dibawah ini merupakan hasil eksekusi dari Simulasi :

```

0:  proc - (:root:) creates proc  0 (:init:)
1:  proc  0 (:init:) creates proc  1 (D)
1:  proc  0 (:init:) line 132 "pan_in" (state 6)  [(run D(q[0],q[1],1))]
2:  proc  0 (:init:) creates proc  2 (D)
2:  proc  0 (:init:) line 135 "pan_in" (state 2)  [(run D(q[1],q[2],2))]
3:  proc  0 (:init:) creates proc  3 (D)
3:  proc  0 (:init:) line 136 "pan_in" (state 3)  [(run D(q[2],q[3],3))]
4:  proc  0 (:init:) creates proc  4 (D)
4:  proc  0 (:init:) line 137 "pan_in" (state 4)  [(run D(q[3],q[4],4))]
5:  proc  0 (:init:) creates proc  5 (Temp)
5:  proc  0 (:init:) line 139 "pan_in" (state 5)  [(run Temp(q[4],q[0],5))]
MSC: 5
6:  proc  5 (Temp) line  78 "pan_in" (state 1)    [printf('MSC:
%d\\n',registernumber)]
MSC: 4
7:  proc  4 (D) line  20 "pan_in" (state 1) [printf('MSC:
%d\\n',registernumber)]
8:  proc  5 (Temp) line  79 "pan_in" (state -)    [values: 1!goshift,5]
8:  proc      5 (Temp) line      79 "pan_in" (state  2)
[out!goshift,registernumber]
MSC: 3
9:  proc  3 (D) line  20 "pan_in" (state 1) [printf('MSC:
%d\\n',registernumber)]
10: proc  5 (Temp) line 127 "pan_in" (state 47)  [.(goto)]
11: proc  3 (D) line  21 "pan_in" (state -) [values: 4!goshift,3]
11: proc  3 (D) line  21 "pan_in" (state 2) [out!goshift,registernumber]
12: proc  3 (D) line  69 "pan_in" (state 47)  [.(goto)]
MSC: 1
13: proc  1 (D) line  20 "pan_in" (state 1) [printf('MSC:
%d\\n',registernumber)]
MSC: 2
14: proc  2 (D) line  20 "pan_in" (state 1) [printf('MSC:
%d\\n',registernumber)]
15: proc  1 (D) line  21 "pan_in" (state -) [values: 2!goshift,1]
15: proc  1 (D) line  21 "pan_in" (state 2) [out!goshift,registernumber]
16: proc  1 (D) line  69 "pan_in" (state 47)  [.(goto)]

```

Pemodelan Shift Register

```
17: proc 2 (D) line 21 "pan_in" (state -) [values: 3!goshift,2]
17: proc 2 (D) line 21 "pan_in" (state 2) [out!goshift,registernumber]
18: proc 2 (D) line 69 "pan_in" (state 47) [.(goto)]
19: proc 3 (D) line 23 "pan_in" (state -) [values: 3?goshift,2]
19: proc 3 (D) line 22 "pan_in" (state 46) [in?goshift,nr]
20: proc 2 (D) line 23 "pan_in" (state -) [values: 2?goshift,1]
20: proc 2 (D) line 22 "pan_in" (state 46) [in?goshift,nr]
21: proc 4 (D) line 21 "pan_in" (state -) [values: 5!goshift,4]
21: proc 4 (D) line 21 "pan_in" (state 2) [out!goshift,registernumber]
22: proc 3 (D) line 24 "pan_in" (state 16) [(Active)]
23: proc 1 (D) line 23 "pan_in" (state -) [values: 1?goshift,5]
23: proc 1 (D) line 22 "pan_in" (state 46) [in?goshift,nr]
24: proc 4 (D) line 69 "pan_in" (state 47) [.(goto)]
25: proc 1 (D) line 24 "pan_in" (state 16) [(Active)]
26: proc 5 (Temp) line 81 "pan_in" (state -) [values: 5?goshift,4]
26: proc 5 (Temp) line 80 "pan_in" (state 46) [in?goshift,nr]
27: proc 4 (D) line 23 "pan_in" (state -) [values: 4?goshift,3]
27: proc 4 (D) line 22 "pan_in" (state 46) [in?goshift,nr]
28: proc 2 (D) line 24 "pan_in" (state 16) [(Active)]
29: proc 3 (D) line 26 "pan_in" (state 12) [((nr!=maximum))]
30: proc 2 (D) line 26 "pan_in" (state 12) [((nr!=maximum))]
31: proc 5 (Temp) line 82 "pan_in" (state 16) [(Active)]
32: proc 3 (D) line 28 "pan_in" (state -) [values: 4!readyshift,2]
32: proc 3 (D) line 28 "pan_in" (state 6) [out!readyshift,nr]
33: proc 2 (D) line 28 "pan_in" (state -) [values: 3!readyshift,1]
33: proc 2 (D) line 28 "pan_in" (state 6) [out!readyshift,nr]
34: proc 2 (D) line 29 "pan_in" (state 7) [neighbourR = nr]
35: proc 3 (D) line 29 "pan_in" (state 7) [neighbourR = nr]
36: proc 4 (D) line 24 "pan_in" (state 16) [(Active)]
37: proc 3 (D) line 37 "pan_in" (state 13) [.(goto)]
38: proc 3 (D) line 41 "pan_in" (state 17) [.(goto)]
39: proc 2 (D) line 37 "pan_in" (state 13) [.(goto)]
40: proc 3 (D) line 69 "pan_in" (state 47) [.(goto)]
41: proc 4 (D) line 26 "pan_in" (state 12) [((nr!=maximum))]
42: proc 2 (D) line 41 "pan_in" (state 17) [.(goto)]
43: proc 4 (D) line 28 "pan_in" (state -) [values: 5!readyshift,3]
43: proc 4 (D) line 28 "pan_in" (state 6) [out!readyshift,nr]
44: proc 4 (D) line 29 "pan_in" (state 7) [neighbourR = nr]
45: proc 3 (D) line 41 "pan_in" (state -) [values: 3?readyshift,1]
45: proc 3 (D) line 22 "pan_in" (state 46) [in?readyshift,nr]
46: proc 5 (Temp) line 84 "pan_in" (state 12) [((nr!=maximum))]
47: proc 3 (D) line 42 "pan_in" (state 29) [(Active)]
```

Pemodelan Shift Register

```
48: proc 2 (D) line 69 "pan_in" (state 47)      [.(goto)]
49: proc 5 (Temp) line 86 "pan_in" (state -)    [values: 1!readyshift,4]
49: proc 5 (Temp) line 86 "pan_in" (state 6)    [out!readyshift,nr]
50: proc 5 (Temp) line 87 "pan_in" (state 7)    [neighbourR = nr]
51: proc 5 (Temp) line 95 "pan_in" (state 13)   [.(goto)]
52: proc 4 (D) line 37 "pan_in" (state 13)     [.(goto)]
53: proc 5 (Temp) line 99 "pan_in" (state 17)   [.(goto)]
54: proc 4 (D) line 41 "pan_in" (state 17)     [.(goto)]
55: proc 3 (D) line 44 "pan_in" (state 25)     [else]
56: proc 5 (Temp) line 127 "pan_in" (state 47)  [.(goto)]
57: proc 1 (D) line 26 "pan_in" (state 12)     [((nr!=maximum))]
58: proc 4 (D) line 69 "pan_in" (state 47)     [.(goto)]
59: proc 3 (D) line 49 "pan_in" (state 24)     [Active = 0]
60: proc 3 (D) line 51 "pan_in" (state 26)     [.(goto)]
61: proc 4 (D) line 41 "pan_in" (state -) [values: 4?readyshift,2]
61: proc 4 (D) line 22 "pan_in" (state 46)     [in?readyshift,nr]
62: proc 4 (D) line 42 "pan_in" (state 29)     [(Active)]
63: proc 5 (Temp) line 99 "pan_in" (state -)    [values: 5?readyshift,3]
63: proc 5 (Temp) line 80 "pan_in" (state 46)  [in?readyshift,nr]
64: proc 1 (D) line 28 "pan_in" (state -) [values: 2!readyshift,5]
64: proc 1 (D) line 28 "pan_in" (state 6) [out!readyshift,nr]
65: proc 1 (D) line 29 "pan_in" (state 7) [neighbourR = nr]
66: proc 5 (Temp) line 100 "pan_in" (state 29) [(Active)]
67: proc 2 (D) line 41 "pan_in" (state -) [values: 2?readyshift,5]
67: proc 2 (D) line 22 "pan_in" (state 46)     [in?readyshift,nr]
68: proc 2 (D) line 42 "pan_in" (state 29)     [(Active)]
69: proc 5 (Temp) line 102 "pan_in" (state 25) [else]
70: proc 3 (D) line 54 "pan_in" (state 30)     [.(goto)]
71: proc 4 (D) line 44 "pan_in" (state 25)     [else]
72: proc 3 (D) line 69 "pan_in" (state 47)     [.(goto)]
73: proc 1 (D) line 37 "pan_in" (state 13)     [.(goto)]
74: proc 2 (D) line 44 "pan_in" (state 25)     [else]
75: proc 5 (Temp) line 107 "pan_in" (state 24) [Active = 0]
76: proc 2 (D) line 49 "pan_in" (state 24)     [Active = 0]
77: proc 2 (D) line 51 "pan_in" (state 26)     [.(goto)]
78: proc 1 (D) line 41 "pan_in" (state 17)     [.(goto)]
79: proc 5 (Temp) line 109 "pan_in" (state 26) [.(goto)]
80: proc 2 (D) line 54 "pan_in" (state 30)     [.(goto)]
81: proc 2 (D) line 69 "pan_in" (state 47)     [.(goto)]
82: proc 4 (D) line 49 "pan_in" (state 24)     [Active = 0]
83: proc 4 (D) line 51 "pan_in" (state 26)     [.(goto)]
84: proc 4 (D) line 54 "pan_in" (state 30)     [.(goto)]
```

Pemodelan Shift Register

```
85: proc 5 (Temp) line 112 "pan_in" (state 30) [.(goto)]
86: proc 4 (D) line 69 "pan_in" (state 47) [.(goto)]
87: proc 1 (D) line 69 "pan_in" (state 47) [.(goto)]
88: proc 1 (D) line 41 "pan_in" (state -) [values: 1?readysift,4]
88: proc 1 (D) line 22 "pan_in" (state 46) [in?readysift,nr]
89: proc 5 (Temp) line 127 "pan_in" (state 47) [.(goto)]
90: proc 1 (D) line 42 "pan_in" (state 29) [(Active)]
91: proc 1 (D) line 44 "pan_in" (state 25)
    [(((neighbourR>nr)&&(neighbourR>maximum)))]
92: proc 1 (D) line 46 "pan_in" (state 21) [maximum = neighbourR]
93: proc 1 (D) line 47 "pan_in" (state -) [values: 2!goshift,5]
93: proc 1 (D) line 47 "pan_in" (state 22) [out!goshift,neighbourR]
94: proc 1 (D) line 51 "pan_in" (state 26) [.(goto)]
95: proc 1 (D) line 54 "pan_in" (state 30) [.(goto)]
96: proc 1 (D) line 69 "pan_in" (state 47) [.(goto)]
97: proc 2 (D) line 23 "pan_in" (state -) [values: 2?goshift,5]
97: proc 2 (D) line 22 "pan_in" (state 46) [in?goshift,nr]
98: proc 2 (D) line 24 "pan_in" (state 16) [else]
99: proc 2 (D) line 38 "pan_in" (state -) [values: 3!goshift,5]
99: proc 2 (D) line 38 "pan_in" (state 15) [out!goshift,nr]
100: proc 3 (D) line 23 "pan_in" (state -) [values: 3?goshift,5]
100: proc 3 (D) line 22 "pan_in" (state 46) [in?goshift,nr]
101: proc 3 (D) line 24 "pan_in" (state 16) [else]
102: proc 3 (D) line 38 "pan_in" (state -) [values: 4!goshift,5]
102: proc 3 (D) line 38 "pan_in" (state 15) [out!goshift,nr]
103: proc 4 (D) line 23 "pan_in" (state -) [values: 4?goshift,5]
103: proc 4 (D) line 22 "pan_in" (state 46) [in?goshift,nr]
104: proc 2 (D) line 41 "pan_in" (state 17) [.(goto)]
105: proc 3 (D) line 41 "pan_in" (state 17) [.(goto)]
106: proc 2 (D) line 69 "pan_in" (state 47) [.(goto)]
107: proc 3 (D) line 69 "pan_in" (state 47) [.(goto)]
108: proc 4 (D) line 24 "pan_in" (state 16) [else]
109: proc 4 (D) line 38 "pan_in" (state -) [values: 5!goshift,5]
109: proc 4 (D) line 38 "pan_in" (state 15) [out!goshift,nr]
110: proc 5 (Temp) line 81 "pan_in" (state -) [values: 5?goshift,5]
110: proc 5 (Temp) line 80 "pan_in" (state 46) [in?goshift,nr]
111: proc 5 (Temp) line 82 "pan_in" (state 16) [else]
112: proc 4 (D) line 41 "pan_in" (state 17) [.(goto)]
113: proc 5 (Temp) line 96 "pan_in" (state -) [values: 1!goshift,5]
113: proc 5 (Temp) line 96 "pan_in" (state 15) [out!goshift,nr]
114: proc 5 (Temp) line 99 "pan_in" (state 17) [.(goto)]
115: proc 4 (D) line 69 "pan_in" (state 47) [.(goto)]
```

Pemodelan Shift Register

```
116: proc 5 (Temp) line 127 "pan_in" (state 47) [.(goto)]
117: proc 1 (D) line 23 "pan_in" (state -) [values: 1?goshift,5]
117: proc 1 (D) line 22 "pan_in" (state 46) [in?goshift,nr]
118: proc 1 (D) line 24 "pan_in" (state 16) [(Active)]
119: proc 1 (D) line 26 "pan_in" (state 12) [else]
120: proc 1 (D) line 33 "pan_in" (state 9) [assert((nr==5))]
121: proc 1 (D) line 34 "pan_in" (state 10) [know_shifting = 1]
122: proc 1 (D) line 35 "pan_in" (state -) [values: 2!shifting,5]
122: proc 1 (D) line 35 "pan_in" (state 11) [out!shifting,nr]
123: proc 1 (D) line 37 "pan_in" (state 13) [.(goto)]
124: proc 1 (D) line 41 "pan_in" (state 17) [.(goto)]
125: proc 2 (D) line 54 "pan_in" (state -) [values: 2?shifting,5]
125: proc 2 (D) line 22 "pan_in" (state 46) [in?shifting,nr]
126: proc 1 (D) line 69 "pan_in" (state 47) [.(goto)]
127: proc 2 (D) line 55 "pan_in" (state 38) [((nr!=registernumber))]
MSC: SHIFT
128: proc 2 (D) line 57 "pan_in" (state 33) [printf('MSC: SHIFT\\n')]
129: proc 2 (D) line 63 "pan_in" (state 39) [.(goto)]
130: proc 2 (D) line 63 "pan_in" (state 43) [else]
131: proc 2 (D) line 65 "pan_in" (state -) [values: 3!shifting,5]
131: proc 2 (D) line 65 "pan_in" (state 42) [out!shifting,nr]
132: proc 2 (D) line 67 "pan_in" (state 44) [.(goto)]
133: proc 2 (D) line 67 "pan_in" (state 45) [goto :b0]
134: proc 2 (D) line 22 "pan_in" (state 48) [break]
135: proc 3 (D) line 54 "pan_in" (state -) [values: 3?shifting,5]
135: proc 3 (D) line 22 "pan_in" (state 46) [in?shifting,nr]
136: proc 3 (D) line 55 "pan_in" (state 38) [((nr!=registernumber))]
MSC: SHIFT
137: proc 3 (D) line 57 "pan_in" (state 33) [printf('MSC: SHIFT\\n')]
138: proc 3 (D) line 63 "pan_in" (state 39) [.(goto)]
139: proc 3 (D) line 63 "pan_in" (state 43) [else]
140: proc 3 (D) line 65 "pan_in" (state -) [values: 4!shifting,5]
140: proc 3 (D) line 65 "pan_in" (state 42) [out!shifting,nr]
141: proc 4 (D) line 54 "pan_in" (state -) [values: 4?shifting,5]
141: proc 4 (D) line 22 "pan_in" (state 46) [in?shifting,nr]
142: proc 4 (D) line 55 "pan_in" (state 38) [((nr!=registernumber))]
143: proc 3 (D) line 67 "pan_in" (state 44) [.(goto)]
MSC: SHIFT
144: proc 4 (D) line 57 "pan_in" (state 33) [printf('MSC: SHIFT\\n')]
145: proc 3 (D) line 67 "pan_in" (state 45) [goto :b0]
146: proc 3 (D) line 22 "pan_in" (state 48) [break]
147: proc 4 (D) line 63 "pan_in" (state 39) [.(goto)]
```

Pemodelan Shift Register

```
148: proc 4 (D) line 63 "pan_in" (state 43) [else]
149: proc 4 (D) line 65 "pan_in" (state -) [values: 5!shifting,5]
149: proc 4 (D) line 65 "pan_in" (state 42) [out!shifting,nr]
150: proc 5 (Temp) line 112 "pan_in" (state -) [values: 5?shifting,5]
150: proc 5 (Temp) line 80 "pan_in" (state 46) [in?shifting,nr]
151: proc 5 (Temp) line 113 "pan_in" (state 38) [else]
152: proc 4 (D) line 67 "pan_in" (state 44) [.(goto)]
153: proc 4 (D) line 67 "pan_in" (state 45) [goto :b0]
MSC: SHIFT LOOP
154: proc 5 (Temp) line 117 "pan_in" (state 35) [printf('MSC: SHIFT
LOOP\\n')]
155: proc 5 (Temp) line 118 "pan_in" (state 36) [nr_leaders =
(nr_leaders+1)]
156: proc 4 (D) line 22 "pan_in" (state 48) [break]
157: proc 5 (Temp) line 119 "pan_in" (state 37) [assert((nr_leaders==1))]
158: proc 5 (Temp) line 121 "pan_in" (state 39) [.(goto)]
159: proc 5 (Temp) line 121 "pan_in" (state 43) [else]
160: proc 5 (Temp) line 123 "pan_in" (state -) [values: 1!shifting,5]
160: proc 5 (Temp) line 123 "pan_in" (state 42) [out!shifting,nr]
161: proc 1 (D) line 54 "pan_in" (state -) [values: 1?shifting,5]
161: proc 1 (D) line 22 "pan_in" (state 46) [in?shifting,nr]
162: proc 1 (D) line 55 "pan_in" (state 38) [((nr!=registernumber))]
163: proc 5 (Temp) line 125 "pan_in" (state 44) [.(goto)]
MSC: SHIFT
164: proc 1 (D) line 57 "pan_in" (state 33) [printf('MSC: SHIFT\\n')]
165: proc 1 (D) line 63 "pan_in" (state 39) [.(goto)]
166: proc 5 (Temp) line 125 "pan_in" (state 45) [goto :b1]
167: proc 5 (Temp) line 80 "pan_in" (state 48) [break]
168: proc 1 (D) line 63 "pan_in" (state 43) [(know_shifting)]
168: proc 5 (Temp) terminates
168: proc 4 (D) terminates
169: proc 1 (D) line 67 "pan_in" (state 44) [.(goto)]
170: proc 1 (D) line 67 "pan_in" (state 45) [goto :b0]
171: proc 1 (D) line 22 "pan_in" (state 48) [break]
171: proc 3 (D) terminates
171: proc 2 (D) terminates
171: proc 1 (D) terminates
171: proc 0 (:init:) terminates
```