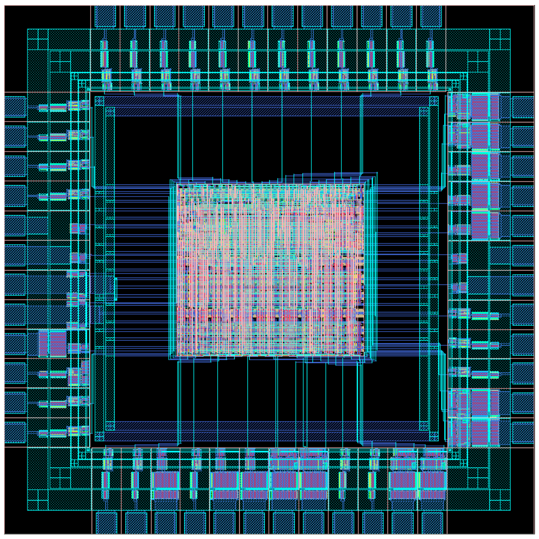


●
●
●

/usr/share/doc/alliance-doc-5.0/tutorial/synthesis/src

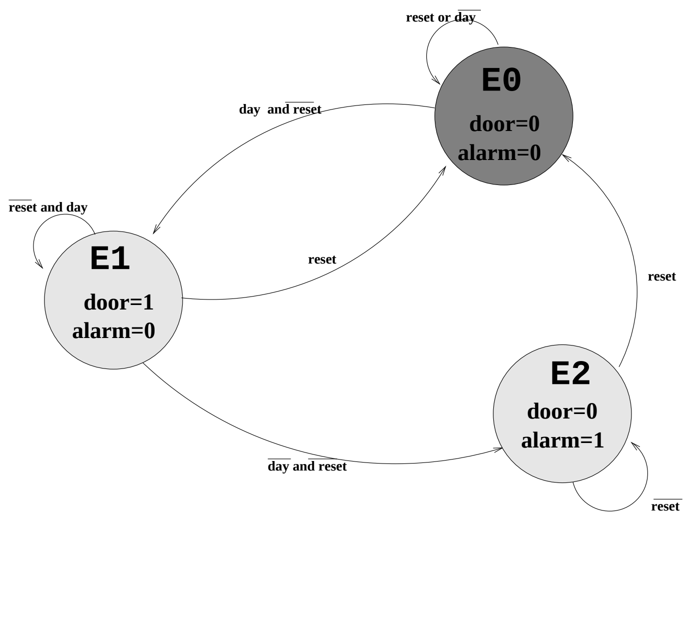
-
-

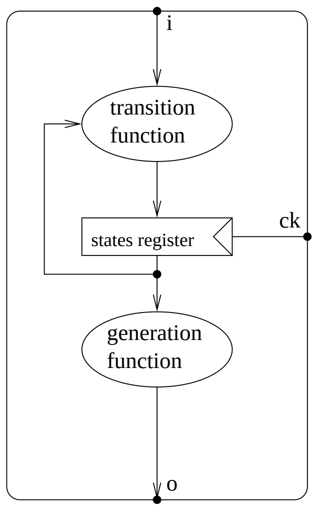
-

-

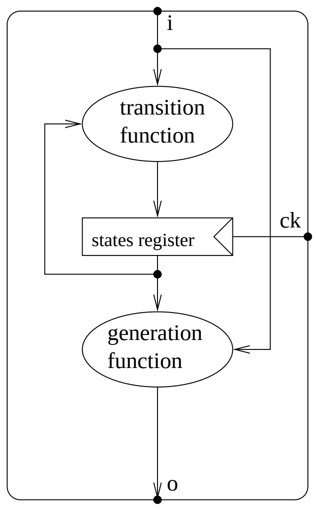
-

●
●
●
●
●

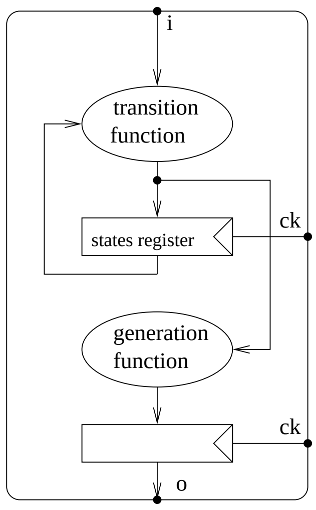

reset or $\overline{\text{day}}$
E0
door=0
alarm=0
day and $\overline{\text{reset}}$
$\overline{\text{reset}}$ and day
E1
door=1
alarm=0
reset
reset
E2
door=0
alarm=1
$\overline{\text{day}}$ and reset
$\overline{\text{reset}}$



Moore Automat

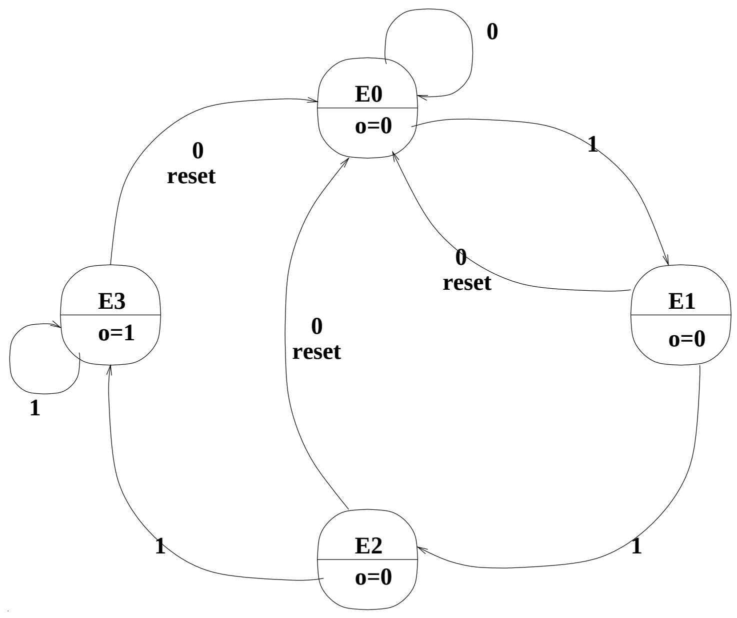


Mealy Automat



Moore Automat optimized on propagation time of o

```
    entity circuit is
          port (
               ck, i, reset, vdd, vss : in bit;
               o : out bit
          );
    end circuit;
    architecture MOORE of circuit is
          type ETAT  _TYPE is (E0, E1, E2, E3);
          signal EF, EP : ETAT  _TYPE;
- - pragma CURRENT  _STATE EP
- - pragma NEXT  _STATE EF
- - pragma CLOCK CK
    begin
          process (EP, i, reset)
          begin
               if (reset='1') then
                    EF<=E0;
               else
                    case EP is
                         when E0 =
                              if (i='1') then
                                   EF <= E1;
                              else
                                   EF <= E0;
                              end if;
                         when E1 =
                              if (i='1') then
                                   EF <= E2;
                              else
                                   EF <= E0;
                              end if;
                         when E2 =
                              if (i='1') then
                                   EF <= E3;
                              else
                                   EF <= E0;
                              end if;
                         when E3 =
                              if (i='1') then
                                   EF <= E3;
                              else
                                   EF <= E0;
                              end if;
                         when others =  assert ('1')
                              report "etat illegal";
                   end case;
               end if;
               case EP is
                    when E0 =
                         o <= '0' ;
                    when E1 =
                         o <= '0' ;
                    when E2 =
                         o <= '0' ;
                    when E3 =
                         o <= '1' ;
                    when others =  assert ('1')
                         report "etat illegal";
              end case;
          end process;
          process(ck)
          begin
               if (ck='1' and not ck'stable) then
                    EP <= EF;
               end if;
          end process;
    end MOORE;
```



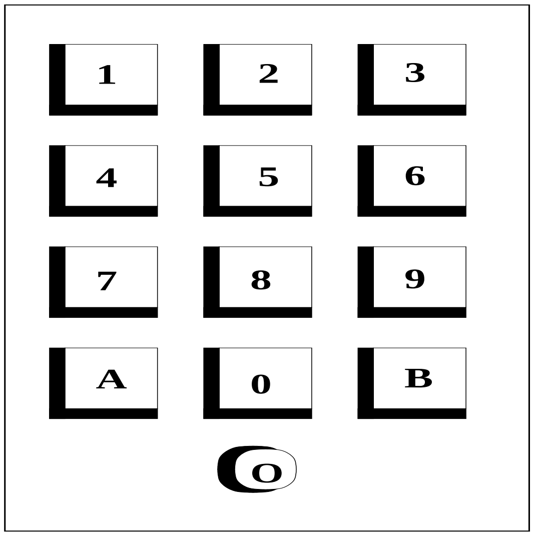
•

•

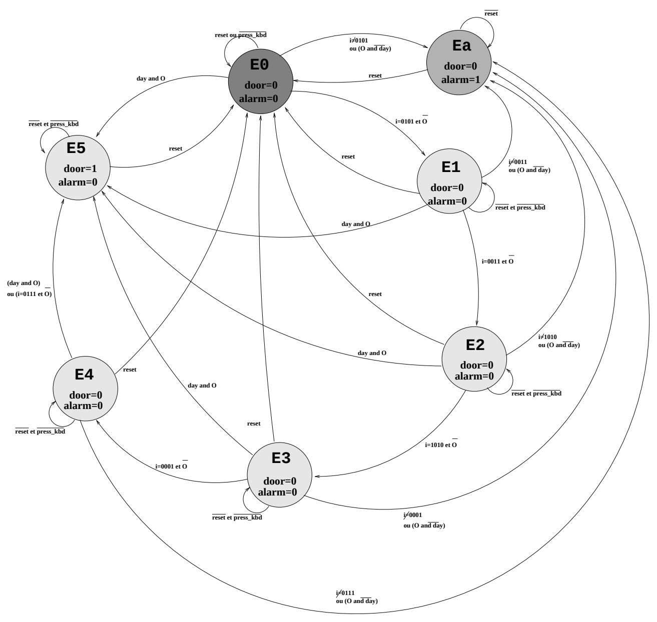
```
> syf -CEV -a <fsm_source>
```

•

•


1
2
3
4
5
6
7
8
9
A
0
B
O

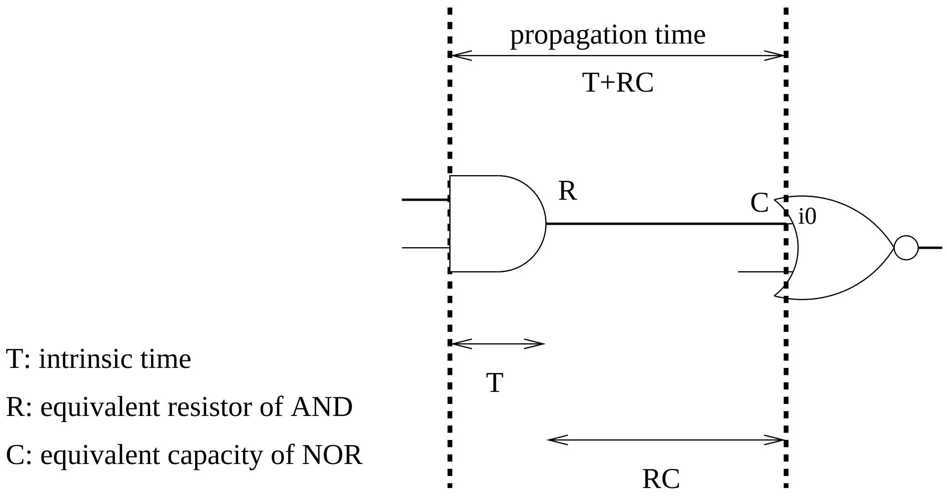
-
-
-
-
-
-
-
-
-


E0
door=0
alarm=0
Ea
door=0
alarm=1
E1
door=0
alarm=0
E2
door=0
alarm=0
E3
door=0
alarm=0
E4
door=0
alarm=0
E5
door=1
alarm=0
$\overline{\text{reset}}$
reset ou $\overline{\text{press\_kbd}}$
i≠0101 ou (O and $\overline{\text{day}}$)
reset
day and O
i=0101 et $\overline{\text{O}}$
$\overline{\text{reset}}$ et $\overline{\text{press\_kbd}}$
i≠0011 ou (O and $\overline{\text{day}}$)
i=0011 et $\overline{\text{O}}$
i≠1010 ou (O and $\overline{\text{day}}$)
i=1010 et $\overline{\text{O}}$
i≠0001 ou (O and $\overline{\text{day}}$)
i=0001 et $\overline{\text{O}}$
(day and O) ou (i=0111 et $\overline{\text{O}}$)
i≠0111 ou (O and $\overline{\text{day}}$)

```
> syf -CEV -a <fsm_source>
```

•

•


propagation time
T+RC
R
C
i0
T
RC
T: intrinsic time
R: equivalent resistor of AND
C: equivalent capacity of NOR

```
> boog <vbe_source>
```

```
>xsch -I vst -l <vst_source>
```

```
>boom -V <vbe_source> <vbe_destination>
```

```
>loon <vst_source> <vst_destination> <lax_param>
```

```
>flatbeh <vst_source> <vbe_dest>
```

```
>proof -d <vbe_origin> <vbe_dest>
```

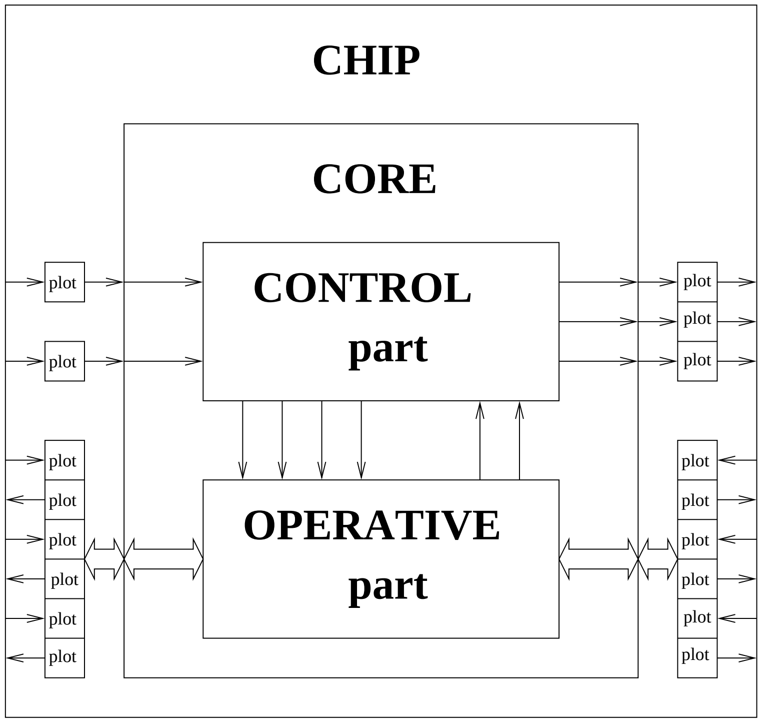
```
>scapin -VRB <vst_source> <path_file> <vst_dest>
```

```
BEGIN_PATH_REG

cs_0
cs_1
cs_2
END_PATH_REG

BEGIN_CONNECTOR

SCAN_IN      scin
SCAN_OUT     scout
SCAN_TEST    test
END_CONNECTOR
```

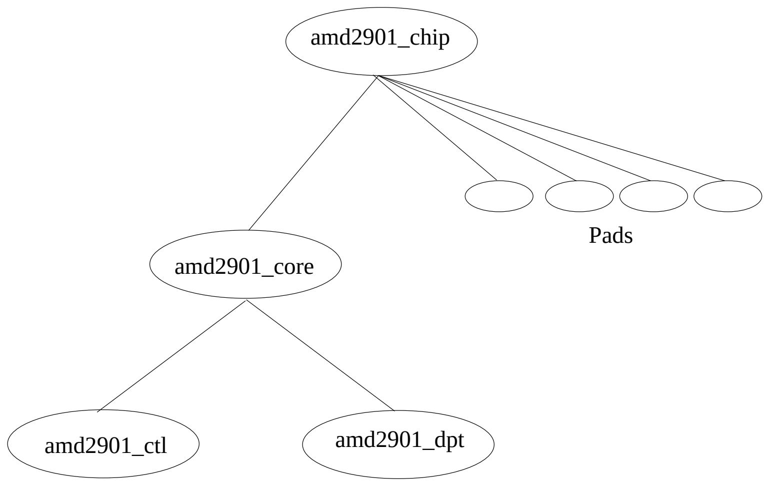
•

●
●
●
●
●

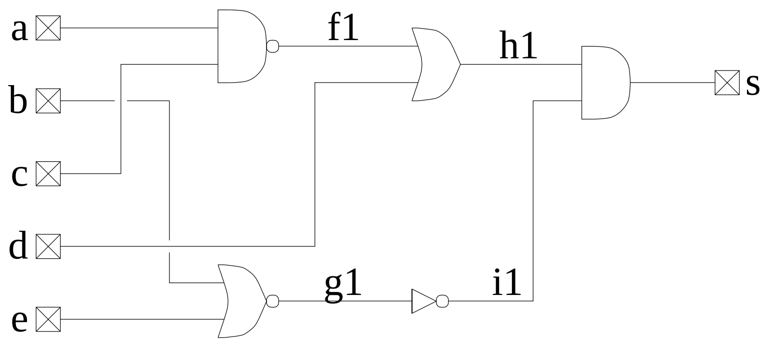
●
●


CHIP
CORE
CONTROL part
OPERATIVE part
plot

-
-


amd2901_chip
Pads
amd2901_core
amd2901_ctl
amd2901_dpt

-
-
-
-
-
-
-
-
-



```
#include <genlib.h>
    main()
      {
       GENLIB_DEF_LOFIG("circuit");

       /* Connectors declaration */
       GENLIB_LOCON("a",IN,"a1");
       GENLIB_LOCON("b",IN,"b1");
       GENLIB_LOCON("c",IN,"c1");
       GENLIB_LOCON("d",IN,"d1");
       GENLIB_LOCON("e",IN,"e1");
       GENLIB_LOCON("s",OUT,"s1");

       GENLIB_LOCON("vdd",IN,"vdd");
       GENLIB_LOCON("vss",IN,"vss");

       /* Combinatorial gates instanciation */
       GENLIB_LOINS("na2_x1","nand2","a1","c1","f1","vdd","vss",0);
       GENLIB_LOINS("no2_x1","nor2","b1","e1","g1","vdd","vss",0);
       GENLIB_LOINS("o2_x2","or2","d1","f1","h1","vdd","vss",0);
       GENLIB_LOINS("inv_x1","inv","g1","i1","vdd","vss",0);
       GENLIB_LOINS("a2_x2","and2","h1","i1","s1","vdd","vss",0);

       /* Save of the figure */
       GENLIB_SAVE_LOFIG();
       exit(0);
      }
```

```
> genlib circuit
```

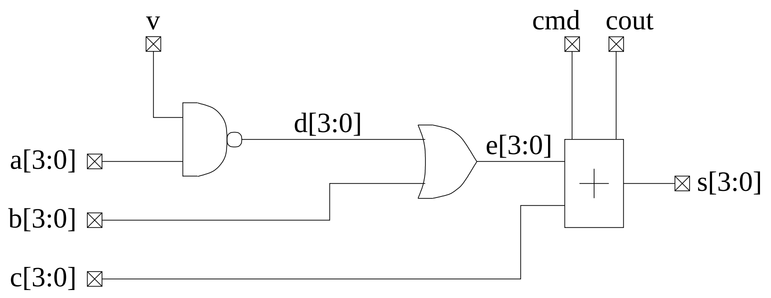
```
amd2901_ctl C
amd2901_dpt C
```

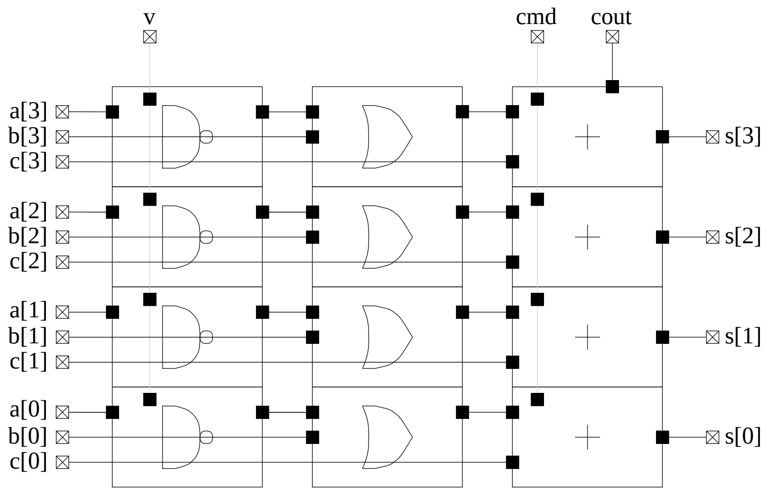
```
> asimut amd2901_chip pattern result
```

```
> genlib amd2901_ctl
```

```
> asimut -zerodelay amd2901_chip vecteurs result
```

```
> asimut amd2901_chip pattern result
```

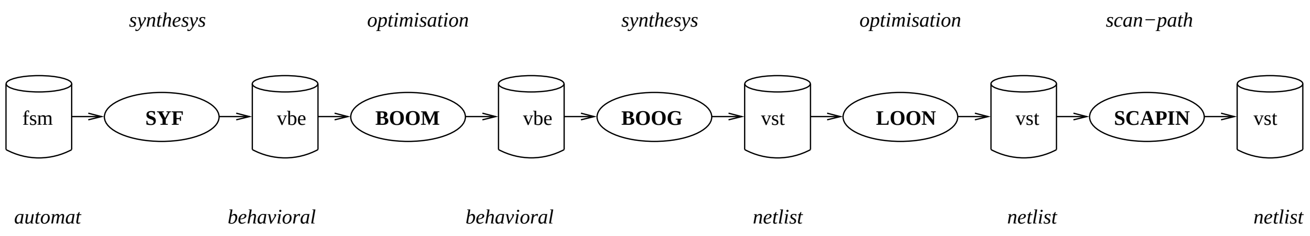

v
cmd
cout
d[3:0]
e[3:0]
a[3:0]
b[3:0]
c[3:0]
s[3:0]


v
cmd
cout
a[3]
b[3]
c[3]
a[2]
b[2]
c[2]
a[1]
b[1]
c[1]
a[0]
b[0]
c[0]
s[3]
s[2]
s[1]
s[0]

```
#include <genlib.h>
    main()
      {
       GENLIB_DEF_LOFIG("data_path");

       /* connectors declaration */
       GENLIB_LOCON("a[3:0]",IN,"a[3:0]");
       GENLIB_LOCON("b[3:0]",IN,"b[3:0]");
       GENLIB_LOCON("c[3:0]",IN,"c[3:0]");
       GENLIB_LOCON("v",IN,"w");
       GENLIB_LOCON("cout",OUT,"ct");
       GENLIB_LOCON("s[3:0]",OUT,"s[3:0]");
       GENLIB_LOCON("cmd",IN,"cmd");
       GENLIB_LOCON("vdd",IN,"vdd");
       GENLIB_LOCON("vss",IN,"vss");

       /* operators creation */
       GENLIB_MACRO(GEN_NAND2, "model_nand2_4bits", F_PLACE, 4, 1);
       GENLIB_MACRO(GEN_OR2, "model_or2_4bits", F_PLACE, 4);
       GENLIB_MACRO(GEN_ADSB2F, "model_add2_4bits", F_PLACE, 4);

       /* operators Instanciation */
       GENLIB_LOINS("model_nand2_4bits", "model_nand2_4bits",
                          "v", "v", "v", "v",
                          "a[3:0]",
                          "d_aux[3:0]",
                          vdd, vss, NULL);
       GENLIB_LOINS("model_or2_4bits", "model_or2_4bits",
                          "d_aux[3:0]",
                          "b[3:0]",
                          "e_aux[3:0]",
                          vdd, vss, NULL);
       GENLIB_LOINS("model_add2_4bits", "model_add2_4bits",
                          "cmd",
                          "cout",
                          "ovr",
                          "e_aux[3:0]",
                          "c[3:0]",
                          "s[3:0]",
                          vdd, vss, NULL);

       /* Save of figure */
       GENLIB_SAVE_LOFIG();
       exit(0);
      }
```

```
> genlib data_path
> genlib amd2901_dpt
```

```
> asimut -zerodelay amd2901_chip pattern result
```



```
target1 : dependence1 dependence2 ....
        #Rq: each command must be preceded by a tabulation
        command_X
        command_Y
            .
            .
            .
```

```
#example of rule for the synthesis
%.vst : %.vbe
      boog $*
```

```
#variables definitions
MY_COPY   = cp -r
MY_NUM    = 42
MY_STRING ="hello"
```

```
#use a variable in a rule

copy:
      ${MY_COPY} digicode.vbe tmp/
```

●

●

●

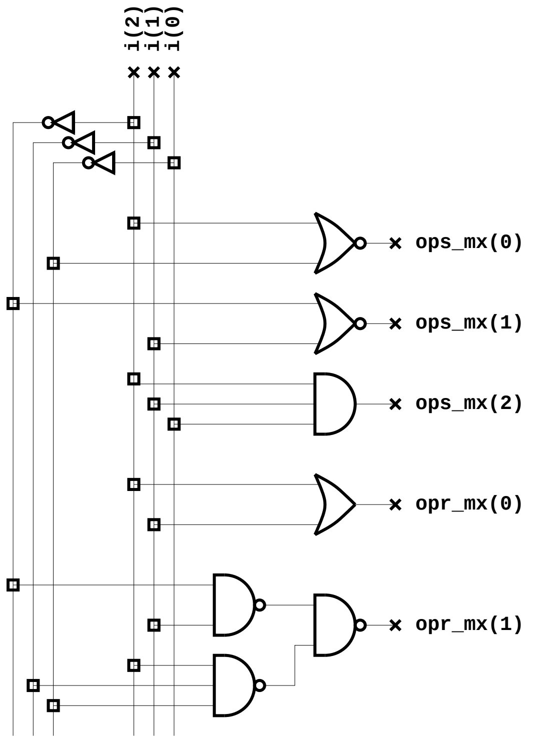
●

● ⌢

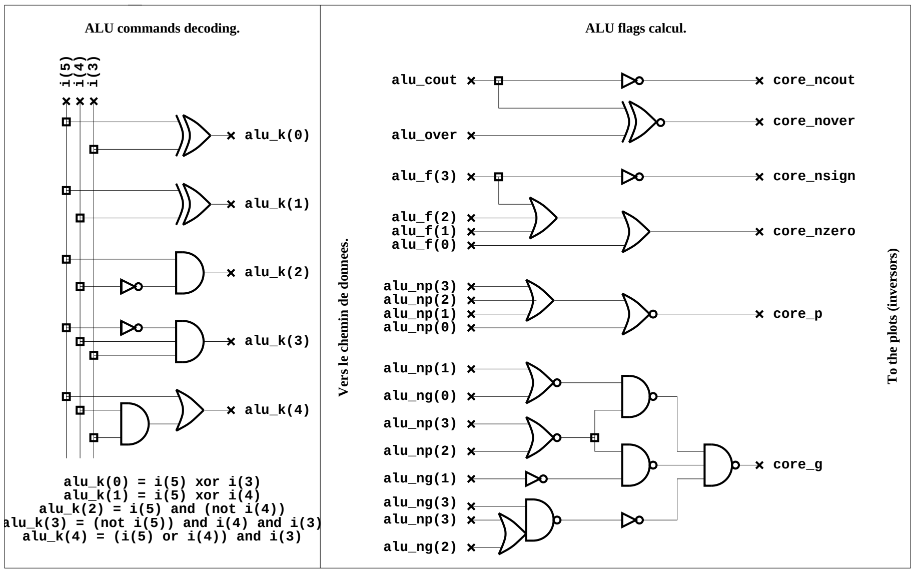
●

●

Decoding of the multiplexers R and S.

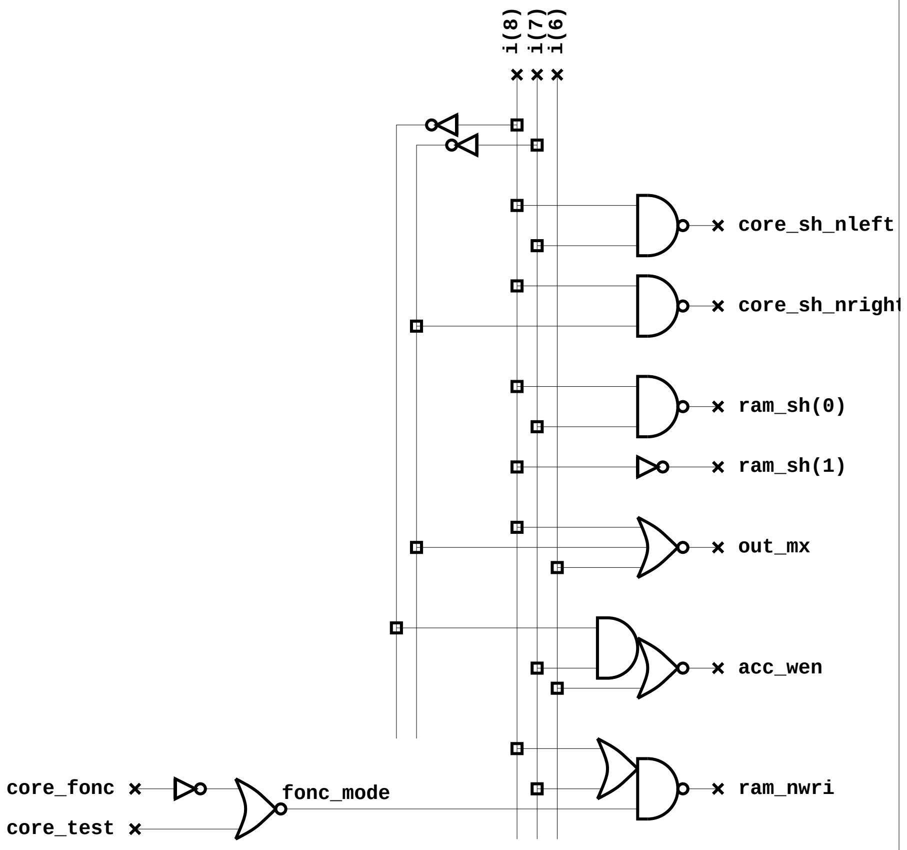


```
ops_mx(0) = (not i(2)) and i(0)
ops_mx(1) = i(2) and (not i(1))
ops_mx(2) = i(2) and i(1) and i(0)

opr_mx(0) = i(2) or i(1)
opr_mx(1) = ((not i(2) and i(1)) or (i(2) and (not i(1)) and (not i(0))
```

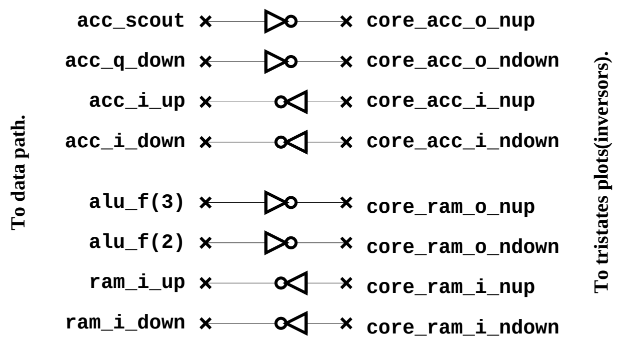


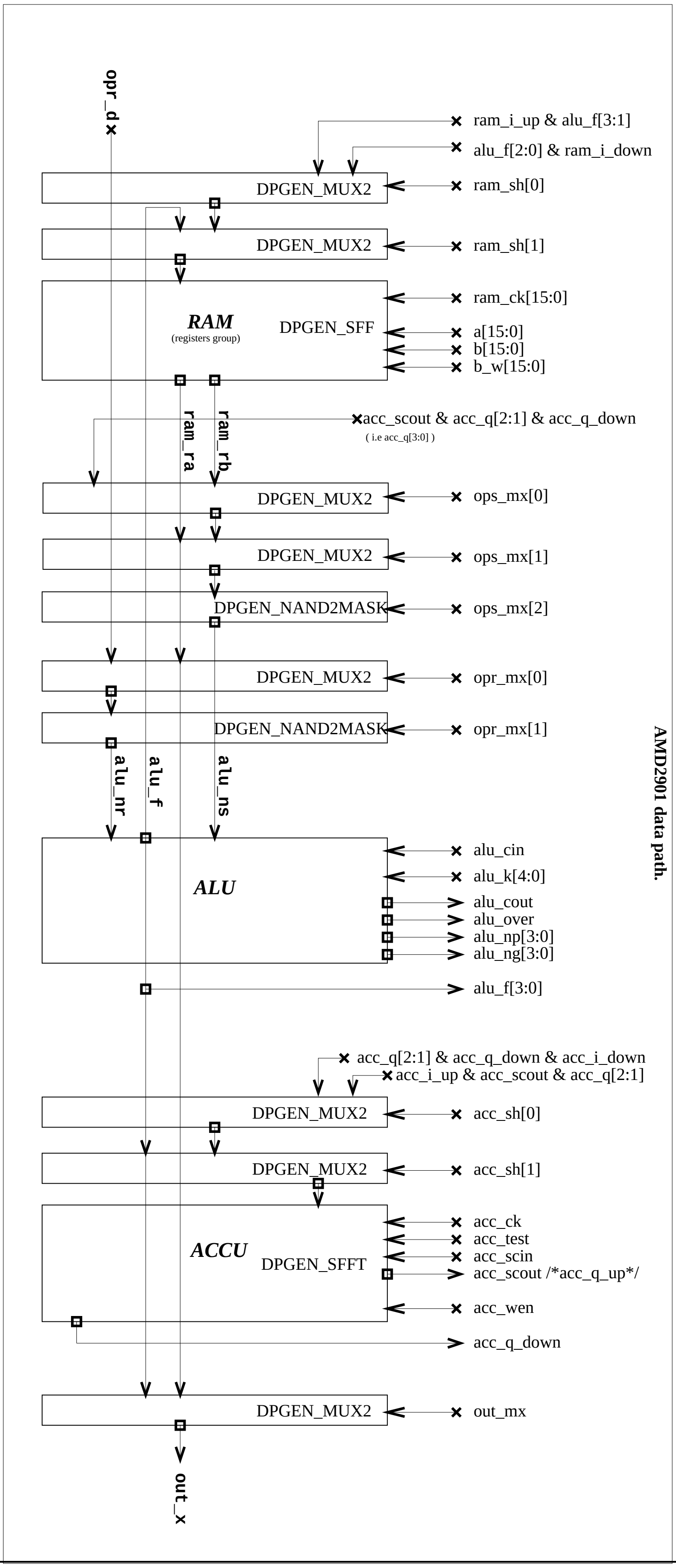
```
alu_k(0) = i(5) xor i(3)
alu_k(1) = i(5) xor i(4)
alu_k(2) = i(5) and (not i(4))
alu_k(3) = (not i(5)) and i(4) and i(3)
alu_k(4) = (i(5) or i(4)) and i(3)
```

shifters control
Write in RAM and ACCU.



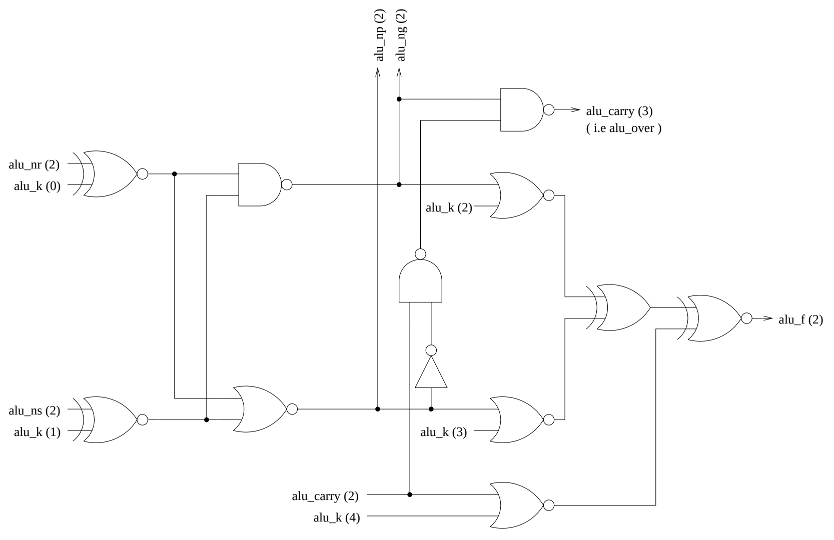
RAM and ACCU inputs/outputs shifters.



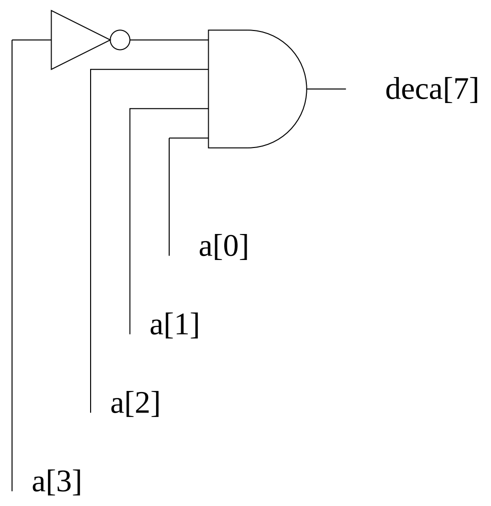


AMD2901 data path.

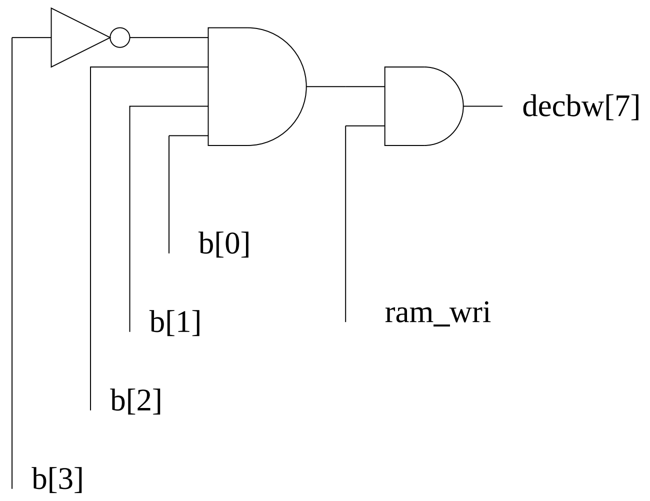
ALU slice 2 representation



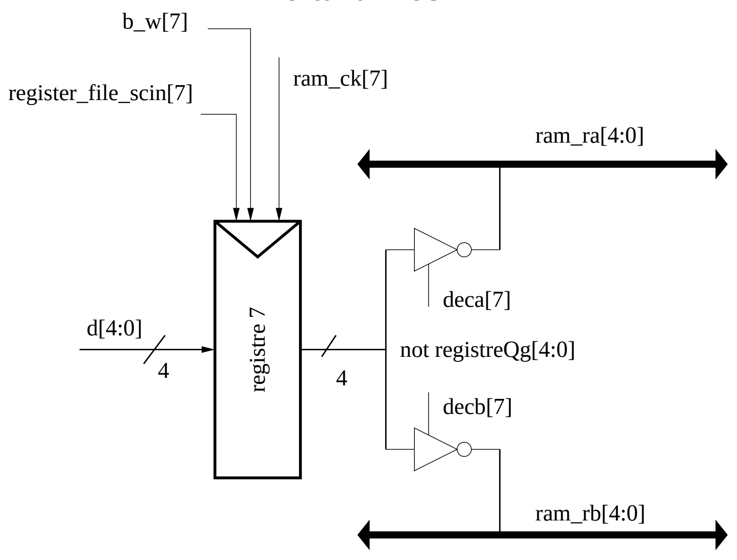
Decoding of the reading command of register 7
BLOCK RAM on port A



write decoding command in register 7 of BLOCK RAM



Slice 7 of BLOCK RAM


b_w[7]
register_file_scin[7]
ram_ck[7]
ram_ra[4:0]
deca[7]
d[4:0]
4
registre 7
4
not registreQg[4:0]
decb[7]
ram_rb[4:0]