EDA PROJECT 2

CSE-215

ZIAD TAREK SALAH

17P3047 CESS

Ziadgyr57@gmail.com

Introduction:

In this phase of our project we'll try to bulid a simple frame decoder chip. Completing on Part 1 of the project where we obtained our Finite State Machine.

In order to simplify design control, Alliance makes use of Makefiles to generate synthesis scripts.

During software development, the *make* utility builds programs from source code using *Makefiles* which specify how to derive the target program. Rules are executed using the unix command *make*. Rules begin with a *dependency line.* Rule format:

- *target*: component files on which the target depends.
- *command_lines*, which define how to get the target.

If any component file is modified, the command lines are run. If ALL component files are not modified and the target file already exists, the commands are not executed. During optimization, we need to *estimate* area, delay and power, with no gates yet. Each variable in a product is called a "literal" (true or complement of variables). Area occupied by logic gates and interconnect is approximated by number of literals (number of transistors). Critical path delay of the longest path through the logic is approximated by maximum number of logic levels. Power consumed by the logic gates is approximated by the complexity of logic. Here we will build a simple frame decoder using the following Alliance Tools:

- syf (Finite State Machine Synthesizer).
- boom (Boolean Minimization).
- boog (Binding And Optimizing On Gates).
- loon (Local Optimization Of Nets).
- xsch (Graphical Netlist Viewer).
- flatbeh (Behavioral From Structural).
- proof (Formal Verification).
- scapin (scan-path insertion).

Encoding Results:

Boom:

Encoding	Literals Number Before Optimization	Literals Number After Optimization
fsmpa	59	33
fsmpj	61	32
fsmpm	61	44
fsmpo	47	29
fsmpr	61	41

Boog:

Encoding	Area (Lamda²)	Delay (ps)
fsmpa	45000	1817
fsmpj	44250	1844
fsmpm	44250	1844
fsmpo	53500	1253
fsmpr	49750	1032

Loon:

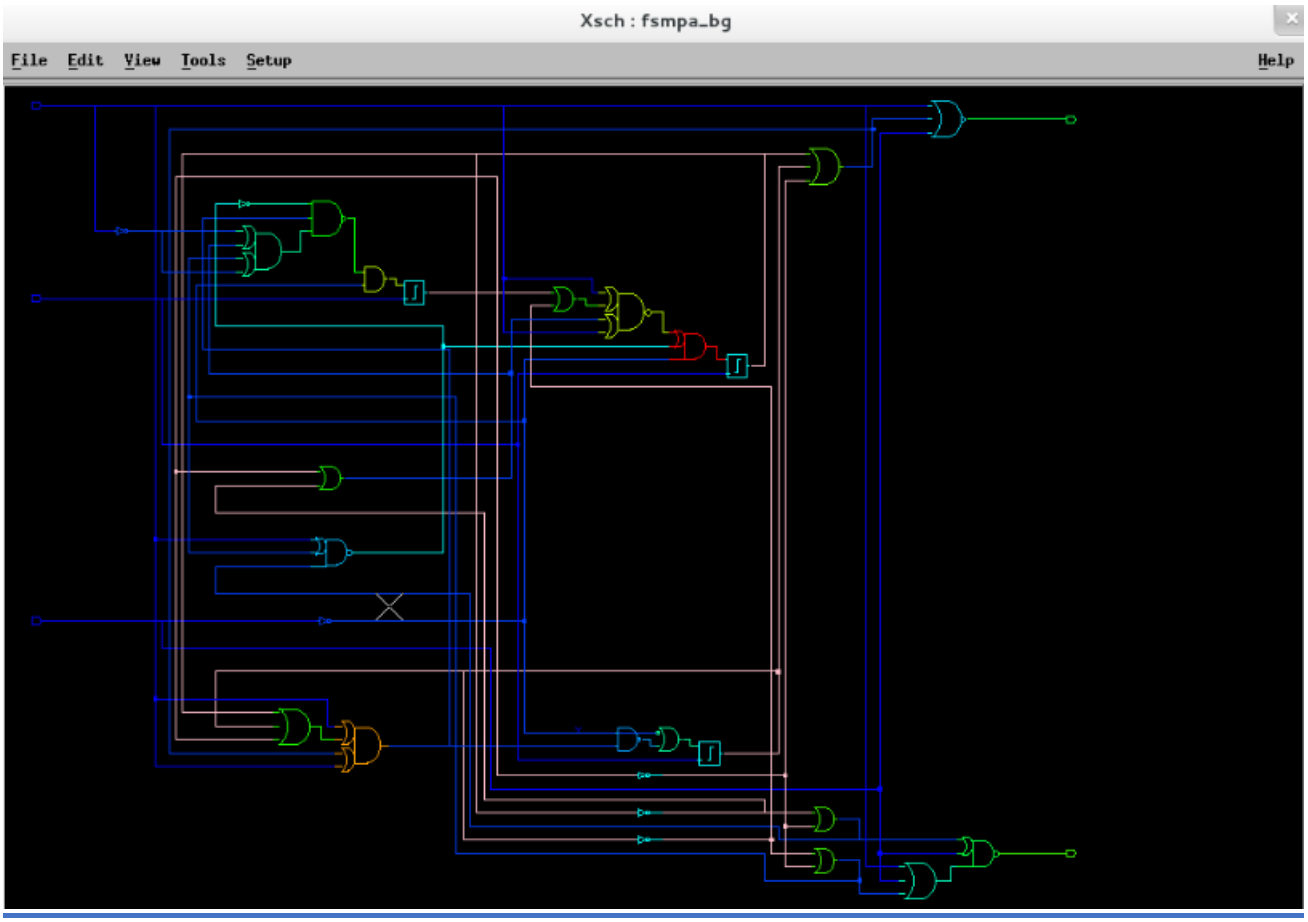
Encoding	New Area (Lamda²)	New Delay (ps)
fsmpa	46250	2296
fsmpj	47500	2358
fsmpm	46500	2379
fsmpo	53500	1251
fsmpr	50750	2522

Chosen Encoding:

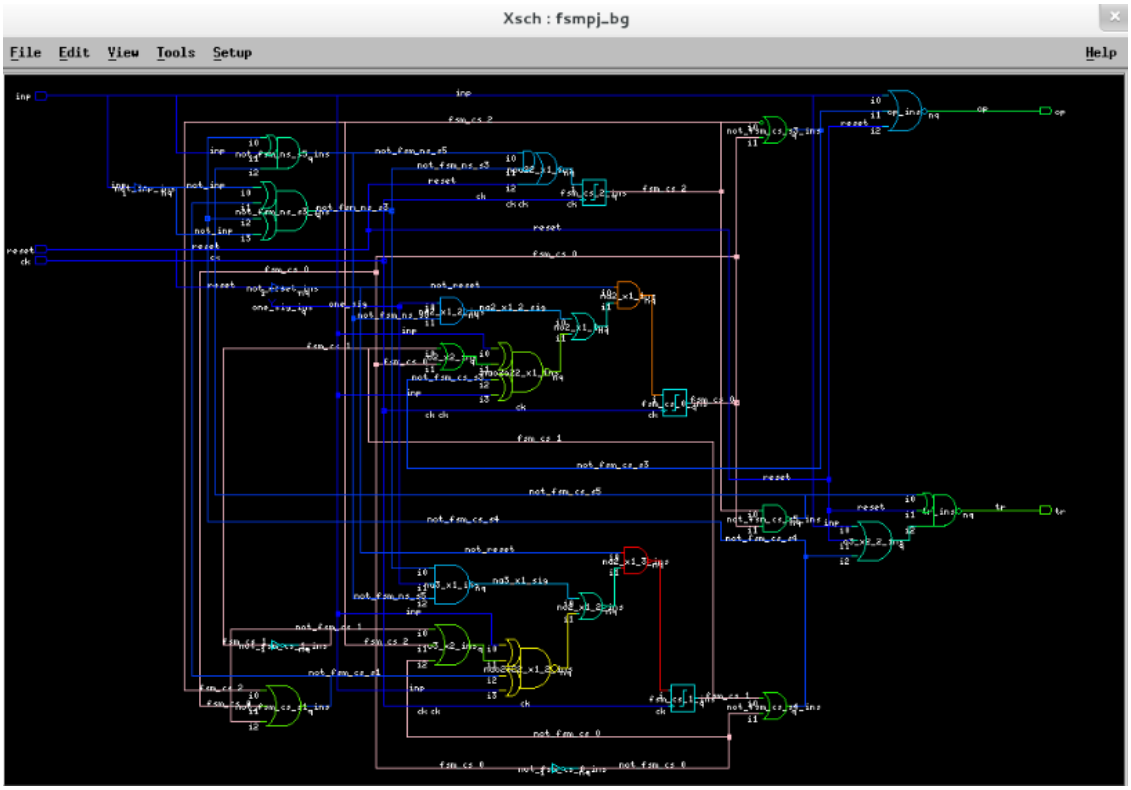
I'd choose -a encoding as it has the least area and the second fewest delay, so I think it is the optimum choice.

Boog Netlists:

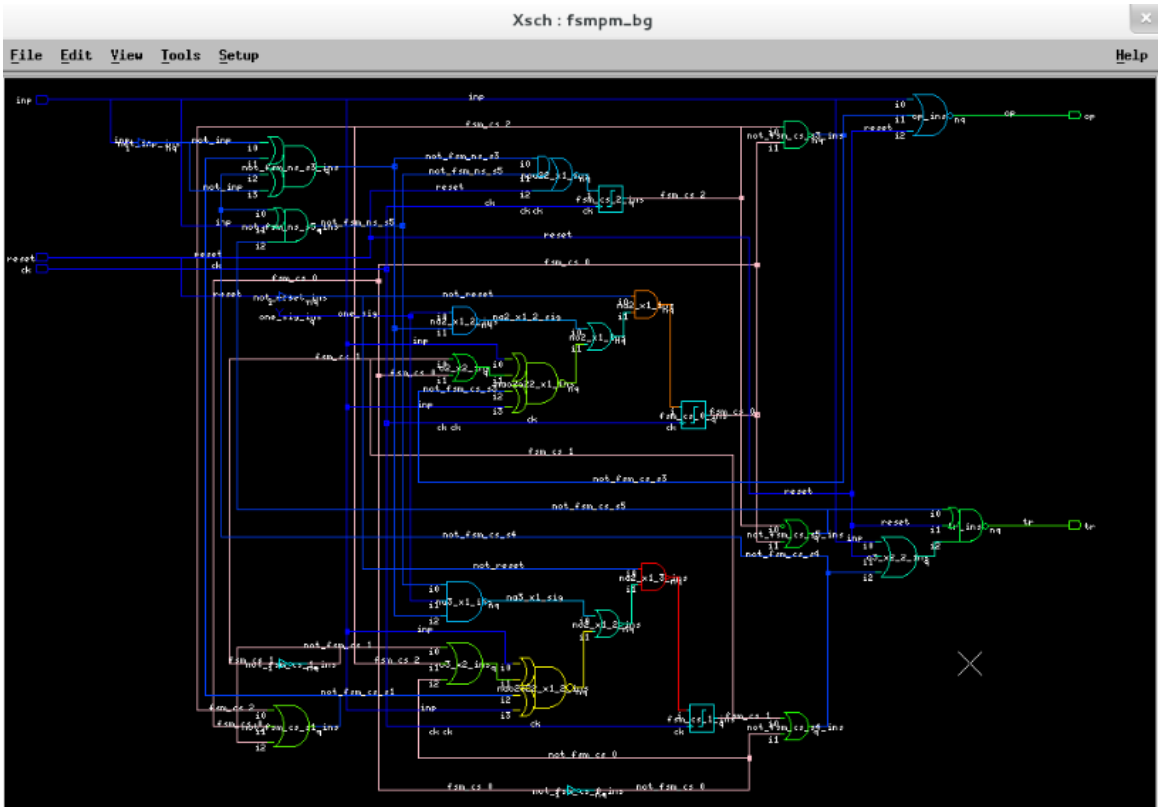
fsmpa_bg



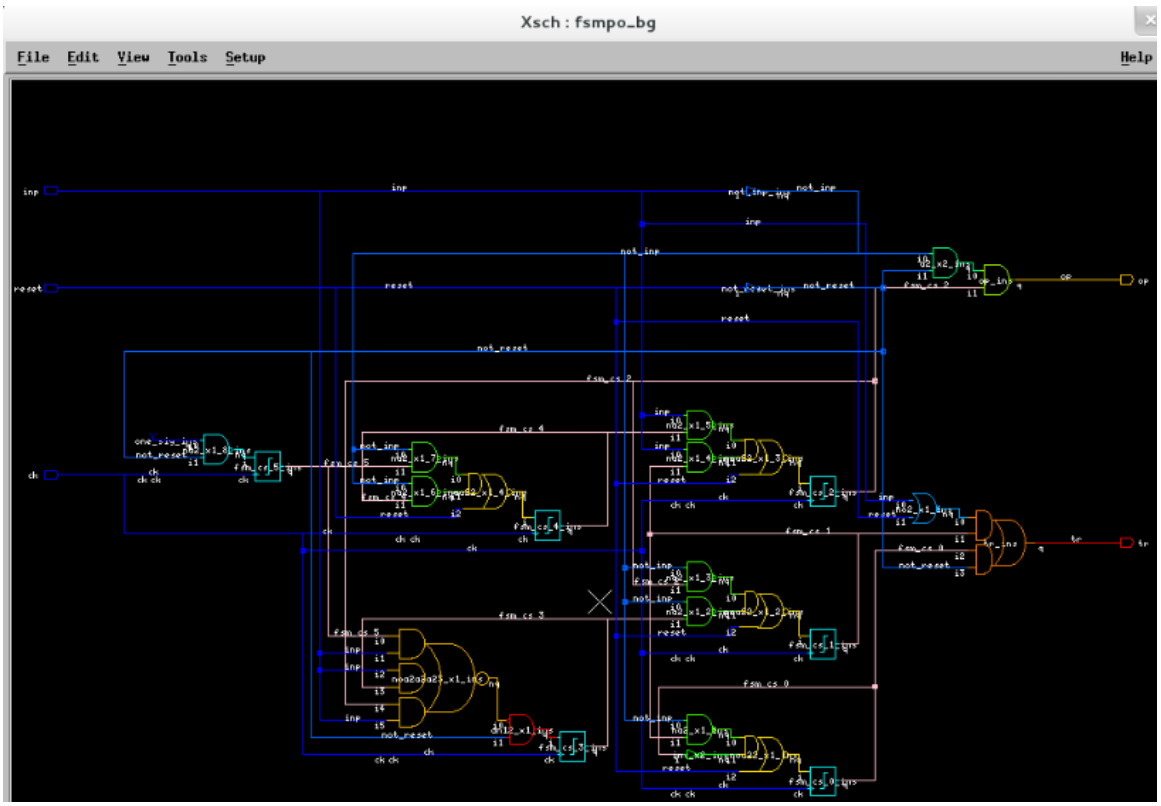
fsmpj_bg



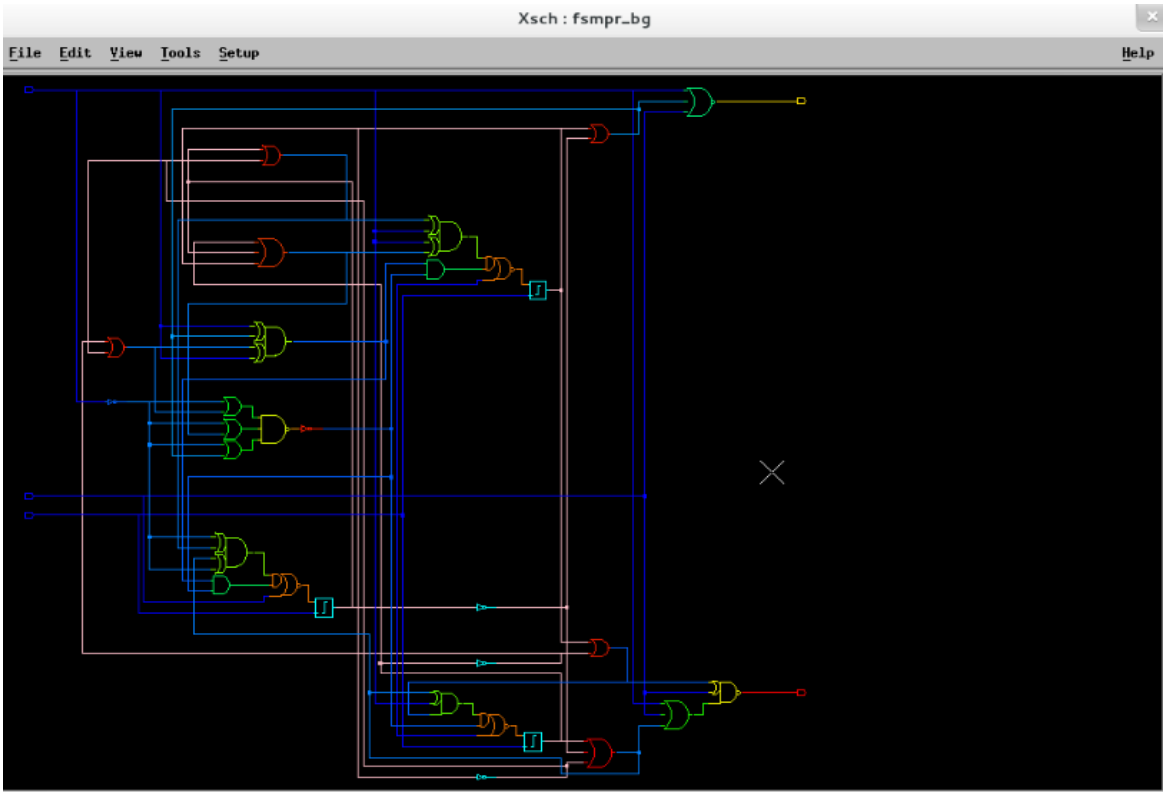
fsmpm_bg



fsmpo_bg

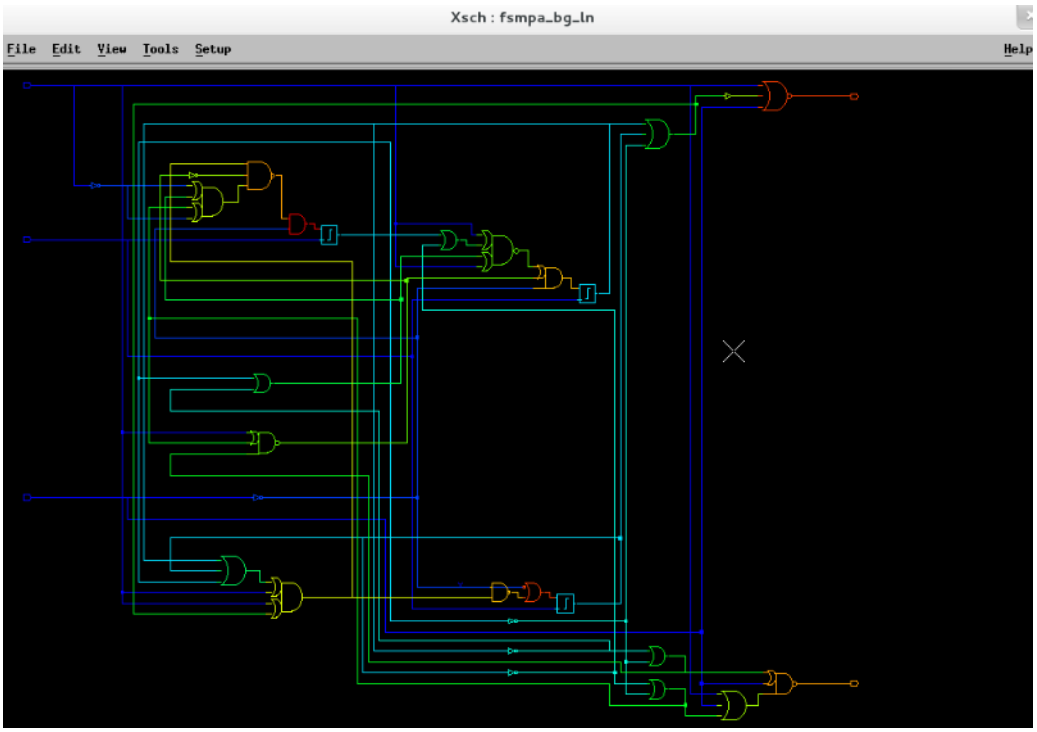


fsmpr_bg

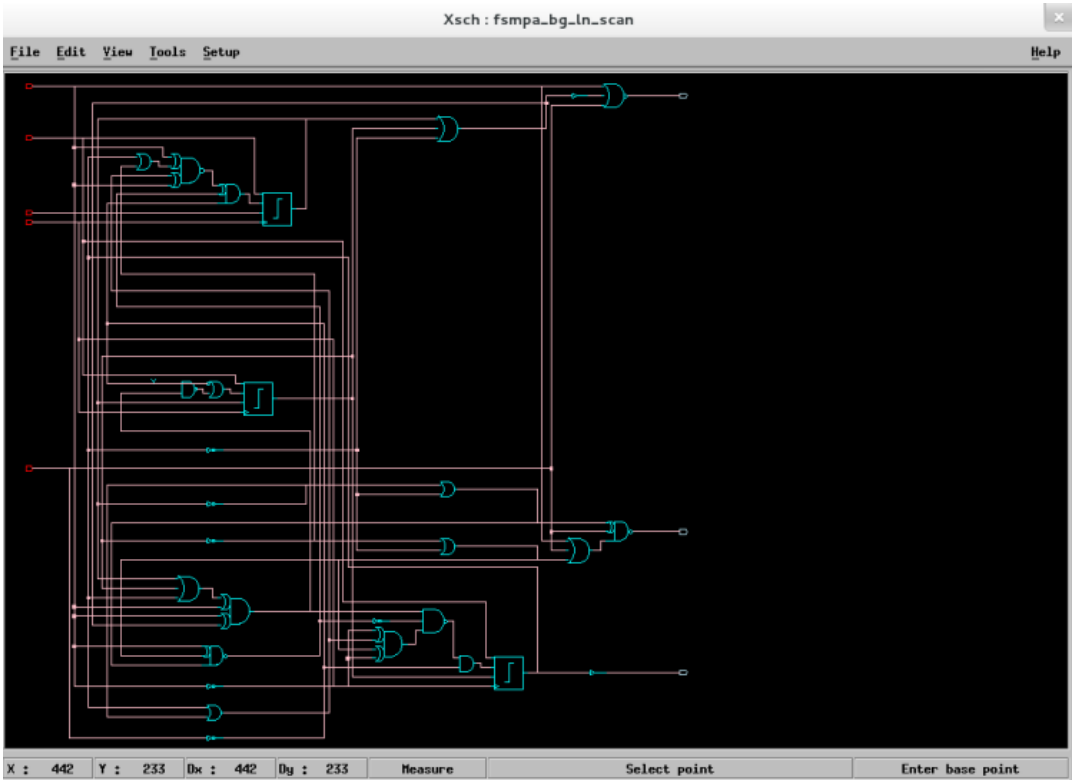


Chosen Encoding Netlist:

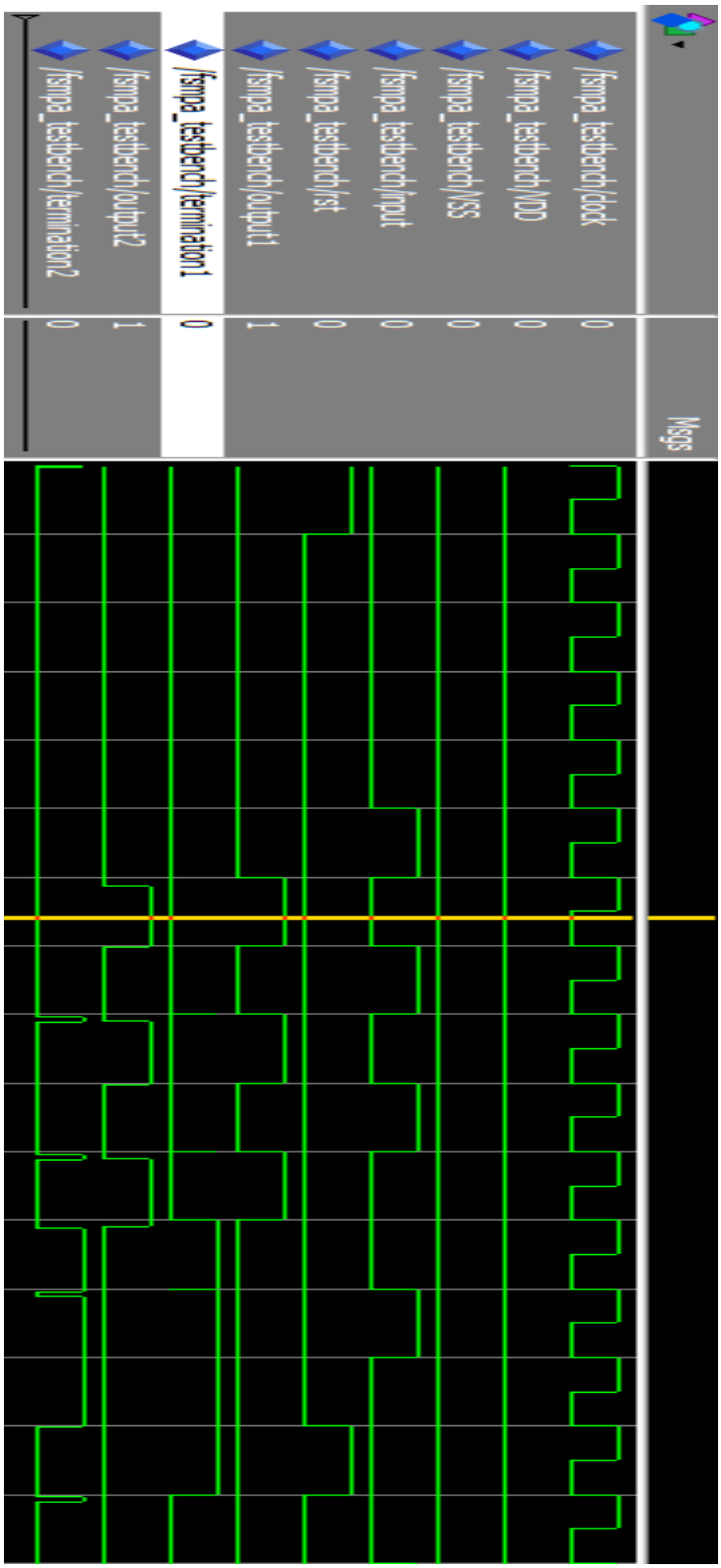
fsmpa_bg_ln



Final Obtained Netlist Critical Path



Delay Simulation Testbench Code And Output:



Code:

```
entity fsmpa_testbench is
end fsmpa_testbench;
architecture test1 of fsmpa_testbench is
component fsm is
        port (
                ck    : in        bit;
                vdd   : in        bit;
                vss   : in        bit;
                inp   : in        bit;
                reset : in        bit;
                op    : out       bit;
                tr    : out       bit
        );
end component fsm;
component fsmpa_bg_ln is
        port (
                 ck    : in      bit;
                 vdd   : in      bit;
                 vss   : in      bit;
                 inp   : in      bit;
                 reset : in      bit;
                 op    : out     bit;
               tr    : out     bit
);
end component fsmpa_bg_ln;

FOR obj1 : fsm USE ENTITY WORK.fsm(archi);
```

```
FOR obj2 : fsmpa_bg_ln USE ENTITY WORK.fsmpa_bg_ln(structural);

SIGNAL clock : bit := '0';
SIGNAL VDD   : bit := '0';
SIGNAL VSS   : bit := '0';
SIGNAL input : bit := '0';
SIGNAL rst : bit := '1';
SIGNAL output1  : bit := '0';
SIGNAL termination1  : bit := '0';
SIGNAL output2  : bit := '0';
SIGNAL termination2  : bit := '0';

constant clk_period : time := 50 ns;
constant sequence1 : bit_vector := "11011010010";

constant sequence2 : bit_vector := "00101010010";

begin

obj1: fsm port map(clock, VDD, VSS, input, rst, output1, termination1);
obj2: fsmpa_bg_ln port map(clock, VDD, VSS, input, rst, output2, termination2);

process is begin
                clock<='1';
                wait for clk_period/2;
                clock<='0';
                wait for clk_period/2;
```

```
END PROCESS;
process is begin
        rst <= '1';
        wait for clk_period;
        rst <= '0';
        wait for clk_period;
        for j in 0 to sequence2'length -1 loop
            wait for clk_period;
            input <= sequence2(j);
        end loop;
        wait for clk_period;
        rst <='1';
        wait for clk_period;
        rst <= '0';
        wait for clk_period;
        input<='1';


WAIT; -- stop process simulation run
END PROCESS;
END ARCHITECTURE test1;
```

Comment:

The above simulation results proof that the output and termination of the structural and behavioral implementations produce the same output. We can see that the structural output is delayed from the behavioral one too, that’s because we’ve taken gates delay into consideration this time.

Scanin Testbench Code And Output:

```
entity fsmpa_testbench is
end fsmpa_testbench;
architecture test1 of fsmpa_testbench is
component fsm is
        port (
                ck    : in       bit;
                vdd   : in       bit;
                vss   : in       bit;
                inp   : in       bit;
                reset : in       bit;
                op    : out      bit;
                tr    : out      bit
        );
end component fsm;
component fsmpa_bg_ln is
        port (
                 ck    : in      bit;
                 vdd   : in       bit;
                 vss   : in      bit;
                 inp   : in      bit;
                 reset : in      bit;
                 op    : out     bit;
               tr    : out     bit
);
```

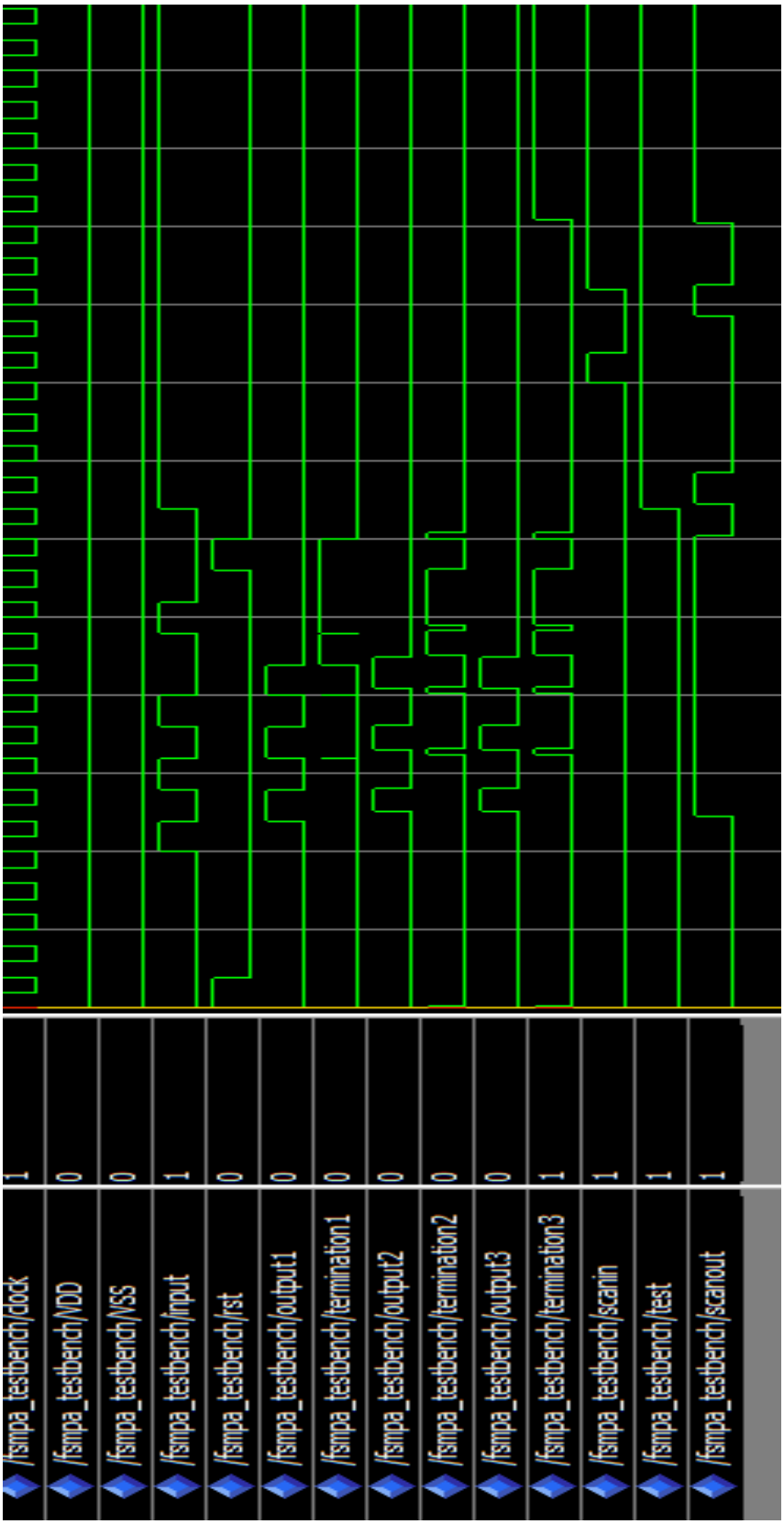
```
end component fsmpa_bg_ln;

component fsmpa_bg_ln_scan is
   port (
      ck      : in      bit;
      vdd     : in      bit;
      vss     : in      bit;
      inp     : in      bit;
      reset   : in      bit;
      op      : out     bit;
      tr      : out     bit;
      scanin  : in      bit;
      test    : in      bit;
      scanout : out     bit
 );
end component fsmpa_bg_ln_scan;

FOR obj1 : fsm USE ENTITY WORK.fsm(archi);
FOR obj2 : fsmpa_bg_ln USE ENTITY WORK.fsmpa_bg_ln(structural);
FOR obj3 : fsmpa_bg_ln_scan USE ENTITY
WORK.fsmpa_bg_ln_scan(structural);

SIGNAL clock : bit := '0';
SIGNAL VDD   : bit := '0';
SIGNAL VSS   : bit := '0';
SIGNAL input : bit := '0';
```

```
SIGNAL rst : bit := '1';
SIGNAL output1  : bit := '0';
SIGNAL termination1  : bit := '0';
SIGNAL output2  : bit := '0';
SIGNAL termination2  : bit := '0';
SIGNAL output3  : bit := '0';
SIGNAL termination3  : bit := '0';
SIGNAL scanin : bit;
SIGNAL test : bit;
SIGNAL scanout : bit ;

constant clk_period : time := 20 ns;
constant sequence1 : bit_vector := "11011010010";

constant sequence2 : bit_vector := "00101010010";

begin

obj1: fsm port map(clock, VDD, VSS, input, rst, output1, termination1);
obj2: fsmpa_bg_ln port map(clock, VDD, VSS, input, rst, output2, termination2);
obj3: fsmpa_bg_ln_scan port map(clock, VDD, VSS, input, rst, output3,
termination3,scanin,test,scanout);

process is begin
        clock<='1';
        wait for clk_period/2;
```

```
        clock<='0';
        wait for clk_period/2;

END PROCESS;

process is begin
        test<='0';
        wait for 32 * (clk_period/2);
        test<='1';
        wait;

END PROCESS;
PROCESS is begin
        wait for 40* (clk_period/2);
        scanin <= '1';
        wait for clk_period;
        scanin <= '0';
        wait for clk_period;
        scanin <= '0';
        wait for clk_period;
        scanin <= '1';
        wait for clk_period;

end process;
process is begin
```

```
        rst <= '1';
        wait for clk_period;
        rst <= '0';
        wait for clk_period;
        for j in 0 to sequence2'length -1 loop
            wait for clk_period;
            input <= sequence2(j);

        end loop;
        wait for clk_period;
        rst <='1';
        wait for clk_period;
        rst <= '0';
        wait for clk_period;
        input<='1';

WAIT; -- stop process simulation run
END PROCESS;
END ARCHITECTURE test1;
```

Simulation Output:



Makefile:

```
#-------fsm-----------------------------------------#

all: fsmpa.vbe \
     fsmpj.vbe \
     fsmpm.vbe \
     fsmpo.vbe \
     fsmpr.vbe
               @echo "<-- Generated"
#-------syf_boom---------------------------------------#
syf_boom: fsmpa_bm.vbe \
           fsmpj_bm.vbe \
           fsmpm_bm.vbe \
          fsmpo_bm.vbe \
           fsmpr_bm.vbe
             @echo "<-- Generated"
#-------syf_boog---------------------------------------#
syf_boog: fsmpa_bg.vst \
           fsmpj_bg.vst \
           fsmpm_bg.vst \
          fsmpo_bg.vst \
           fsmpr_bg.vst
             @echo "<-- Generated"
#-------syf_loon---------------------------------------#
syf_loon: fsmpa_bg_ln.vst \
```

```
            fsmpj_bg_ln.vst \
            fsmpm_bg_ln.vst \
          fsmpo_bg_ln.vst \
            fsmpr_bg_ln.vst
               @echo "<-- Generated"
#-------proof-----------------------------------------#
proof: fsmpa_bg_ln_net.vbe
               @echo "<-- Generated"
#-------ac_scapin_registers---------------------------------------#
DFT: fsmpa_bg_ln_scan.vst
               @echo "<-- Generated"




#-------Finite State Machine Synthesis-----------------#

vhd_to_fsm:
          rename .vhd .fsm *.vhd

fsmpa.vbe: fsmp.fsm
          @echo "    Encoding Synthesis -> fsma.vbe"
          syf -CEV -a fsmp

fsmpj.vbe: fsmp.fsm
```

```
	@echo "    Encoding Synthesis  -> fsmj.vbe"
	syf -CEV -j fsmp

fsmpm.vbe: fsmp.fsm
	@echo "    Encoding Synthesis  -> fsmm.vbe"
	syf -CEV -m fsmp

fsmpo.vbe: fsmp.fsm
	@echo "    Encoding Synthesis  -> fsmo.vbe"
	syf -CEV -o fsmp

fsmpr.vbe: fsmp.fsm
	@echo "    Encoding Synthesis  -> fsmr.vbe"
	syf -CEV -r fsmp

runall: fsmpa.vbe fsmpj.vbe fsmpm.vbe fsmpo.vbe fsmpr.vbe

#-------Boom--------------------------------------------#
%_bm.vbe : %.vbe
	@echo "    Boolean Optimization -> $@"
	boom -V -d 50 $* $*_bm > $*_boom.out
#-------Boog--------------------------------------------#
%_bg.vst: %.vbe paramfile.lax
	@echo "    Logical Synthesis -> $@"
	boog -x 1 -l paramfile $* $*_bg > $*_boog.out
```

```
#-------loon----------------------------------------#
%_ln.vst: %.vst paramfile.lax
	@echo "    Netlist Optimization -> $@"
	loon -x 1 -l paramfile $* $*_ln > $*_loon.out
#-------proof---------------------------------------#
%_bg_ln_net.vbe: %_bg_ln.vst %.vbe
	@echo "    FormalCecking -> $@"
	flatbeh $*_bg_ln $*_bg_ln_net> $*_flatbeh.out
	proof -d $* $*_bg_ln_net > $*_proof.out
#-------DFT-----------------------------------------#
%_scan.vst: %.vst scan.path
	@echo "    scan-path insertion -> $@"
	scapin -VRB $* scan $*_scan > scapin.out



#-------Clean Up-------------------------------------#

clean :
	rm -f  *.vbe *.enc *~
	@echo "Erase all the files generated by the makefile"
```

Paramfile.lax:

#M{2}

#L{2}

#C{

op:100;

tr:100;

}

Scan.path:

BEGIN_PATH_REG

fsm_cs_0_ins

fsm_cs_1_ins

fsm_cs_2_ins

END_PATH_REG

BEGIN_CONNECTOR

SCAN_IN scanin

SCAN_OUT scanout

SCAN_TEST test

END_CONNECTOR