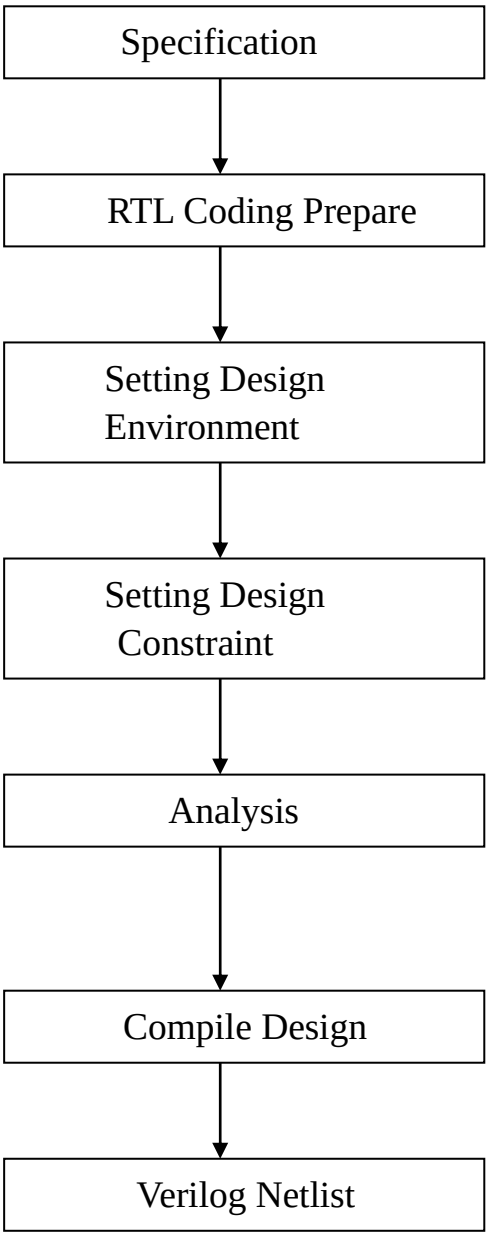
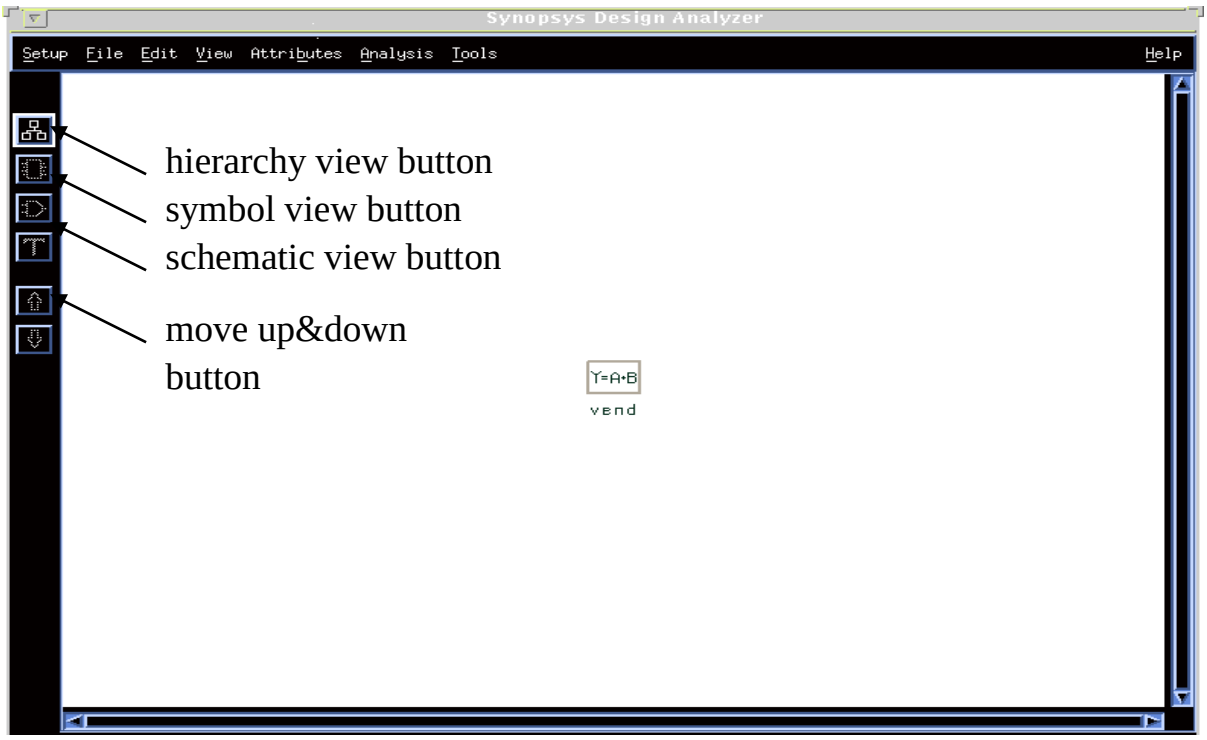
Chapter 6 Logic Synthesis

6.1 What is logic Synthesis

Simply speaking, logic synthesis is the process of converting a high level description of the design into an optimized gate level representation, given a standard cell library and certain design constraints. Automated logic synthesis has significantly reduced time for conversion from high-level design representation to gates. This has allowed designers to spend more time on designing at a high level of representation, because less time is require for converting the design to gate.



6.2 Synthesis Environment



Step1: Read File

File → Read

Use left key click on .v file, then click OK button.

Step2: Setting Design Environment

1. Setting input driving strength

   Select port clk.

   Attributes→Operating Environment→Drive Strength

2. Setting output capacitance loading

   Select all output ports by drag the left key

   Attributes→Operating Environment→Load

3. Setting operating condition

   Attributes→Operating Environment→Operating Conditions

   Click on WCCOM

Step3: Set Design Constraint

1. Specify clock

   Select clk

   Attributes→Clocks→Specify

2.Setting input delay

Select set_time

Attributes→Operating Environment →Input Delay

3.Setting output delay

Select all the output ports

Attributes→Operating Environment→Output Delay

4.Setting area constrains

Attributes→Optimization Constraints→Design Constraints

Step4: Compile Design

1.Compile Design

Select top

Tools→Design Optimization

2. Work with some view commands

View→Recreate

View→Zoom In

View→Zoom out

View→Full View

Step5: Analysis

1.Check the constraints, area, timing.

Select top

Analysis→Report

2.Examine the critical path

Click the schematic button, will show the schematic view of top

Analysis→Highlight→Critical Path

3. Remove highlight

Analysis→Highlight→Clear

4. Generate another version of a timing report

Analysis →Report

Step6: Save File

Save compiled design as db file and verilog netlist.

File→Save as

Step7: Quit Design Analyzer

File→Quit

6.3 Verilog Constructs

Not all constructs can be used when writing a description for a logic synthesis tool. In general, any construct that is used to define a cycle-by-cycle RTL description is acceptable to the logic synthesis tool. A list of constructs that are typically accepted by logic synthesis tools is shown below.

Construct Type	Keyword	Notes
Ports	input , inout, output	
Parameters	parameter	
Module definition	module	
Signals and variables	wire, reg, tri	vetors are allowed
Instantiation	module instances primitive gate instances	eg:mymux (out,i0,i1,s) eg:nand (out,a,b)
Function and tasks	function, task	timing constructs ignored
Procedural	always, if, then, else, case, casez, casex	initial is not supported
Procedural blocks	begin, end, named, blocks, disable	disabling of named blocks allowed
Data flow	assign	delay information is ignored
Loops	for, while, forever	while and forever loops must contain @(posedge clk) or @(negedge clk)

6.4 Verilog Operators

Almost all operators in verilog are allowed for logic synthesis. Only operators such as = = = and ! = = that are related x and z are not allow, because equality with x and z does not have much meaning in logic synthesis. While writing expressions, it is recommended that you use parentheses to group logic the way you want it to appear. If you rely on operator precedence, logic synthesis tools might produce undesirable logic structure.

Operator Type	Operator Symbol	Operation Performed
Arithmetic	*	multiply
	/	divide
	+	add
	--	subtract
	%	modulus
	+	unary plus
	--	unary minus
Logical	!	logic negation
	&&	logic and
	||	logic or
Relational	>	greater than
	<	less than
	> =	greater than or equal
	< =	less than equal
Equality	= =	equality
	! =	inequality
Bit-wise	~	bitwise negation
	&	bitwise and
	|	bitwise or
	^	bitwise ex-or
	^ ~ or ~ ^	bitwise ex-nor
Reduction	&	reduction and
	~&	reduction nand
	|	reduction or
	~ |	reduction nor
	^	reduction ex-or
	^ ~ or ~ ^	reduction ex-nor
Shift	> >	right shift
	< <	left shift
Concatenation	{ }	concatenation
Conditional	? :	conditional

6.5 Basic Design Example

Let us discuss synthesis of two examples to understand each step in the synthesis flow.

6.5.1 4-Bit Magnitude Comparator

A magnitude comparator checks if one number is greater than, equal to, or less than another number. Design a 4_bit magnitude comparator that has the following specifications :

* Output A_gt_B is true if A is greater than B
* Output A_eq_B is true if A is equal than B
* Output A_lt_B is true if A is less than B

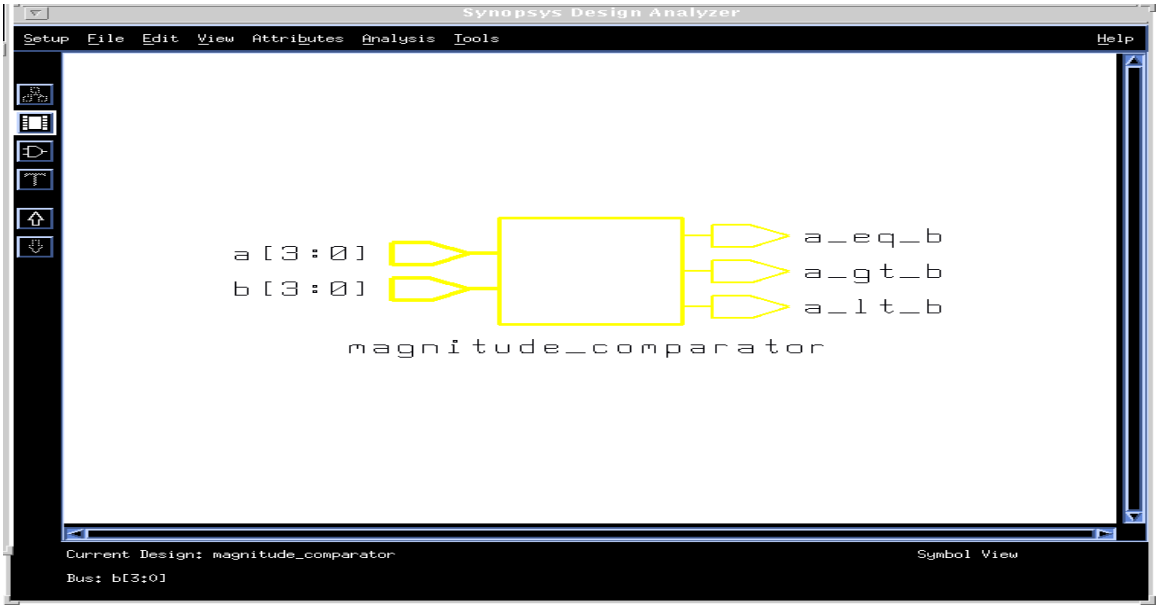
Verilog Code

```
//Module magnitude comparator
module magnitude_comparator(A_gt_B, A_lt_B, A_eq_B,A,B);

//Comparison output
output A_gt_B,A_lt_B,A_eq_B;

//4-bits numbers input
input [3:0] A,B;

assign A_gt_B=(A>B);
assign A_eq_B=(A==B);
assign A_ls_B=(A<B);

endmodule
```

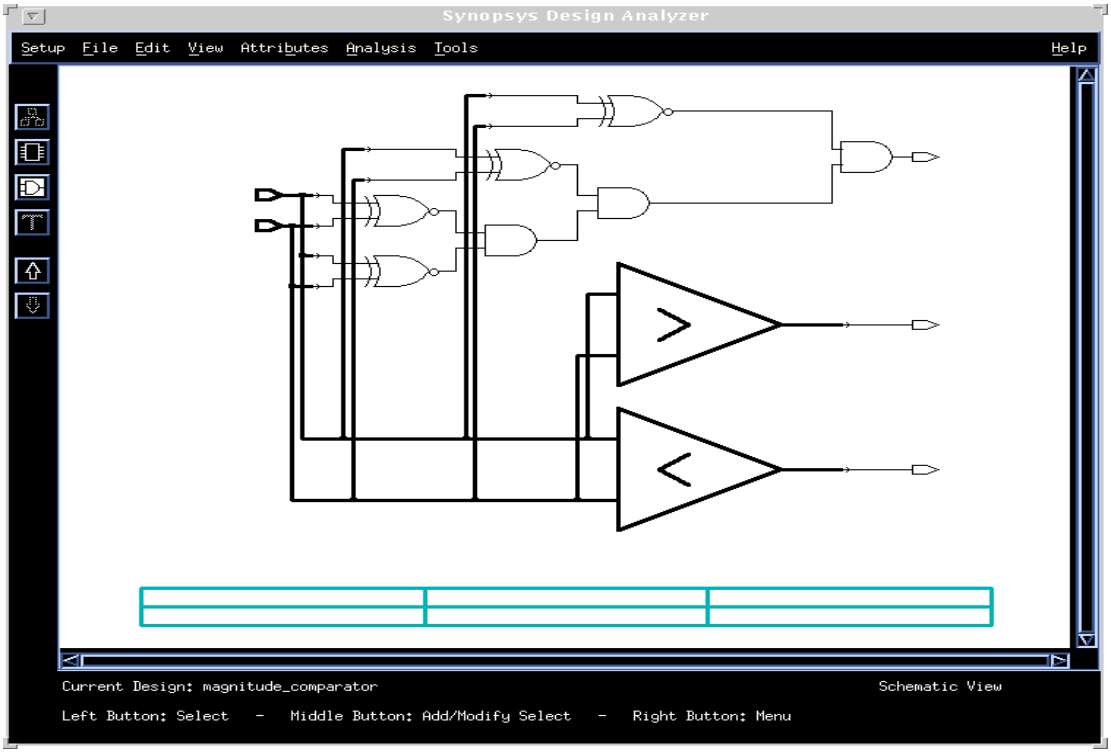
Synthesis Result

The logic synthesis tool produces a final, gate-level description. The schematic for the gate-level circuit is shown below.

Symbol View



Schematic View



Test Stimulus Code

For the magnitude comparator, a sample stimulus file is shown below.

```
module stimulus;
reg [3:0] A,B;
wire A_gt_B,A_eq_B,A_lt_B;
magnitude_comparator m1(A_gt_B, A_lt_B,
A_eq_B,A,B);
initial
begin
$monitor("A=%b,B=%b,A_gt_B=%b,A_eq_B=%b,A_lt_B=%b",
A,B,A_gt_B,A_eq_B,A_lt_B);
end

initial
begin
      A=4'b1111 ; B=4'b1100;
#10 A=4'b1100 ; B=4'b1100;
#10 A=4'b1000 ; B=4'b1100;
#10 A=4'b0111 ; B=4'b0001;

end
```

A=B
A<B
A>B

Simulation Result

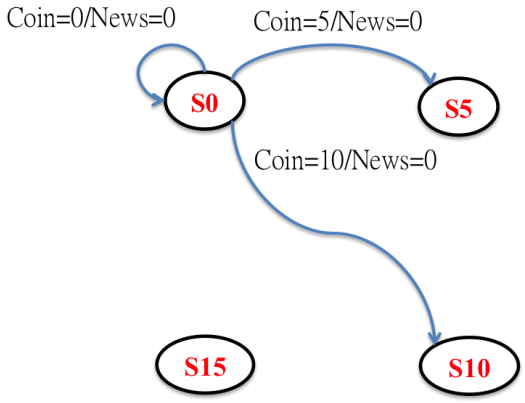
```
    LINIT begins
A=1111,B=1100,A_gt_B=1,A_eq_B=0,A_lt_B=0
    LINIT done

    0 State changes on observable nets.

    Simulation stopped at the end of time 0.
Ready: sim
A=1100,B=1100,A_gt_B=0,A_eq_B=1,A_lt_B=0
A=1000,B=1100,A_gt_B=0,A_eq_B=0,A_lt_B=1
A=0111,B=0001,A_gt_B=1,A_eq_B=0,A_lt_B=0

    26 State changes on observable nets.

    Simulation stopped at the end of time 30.
Ready:
```

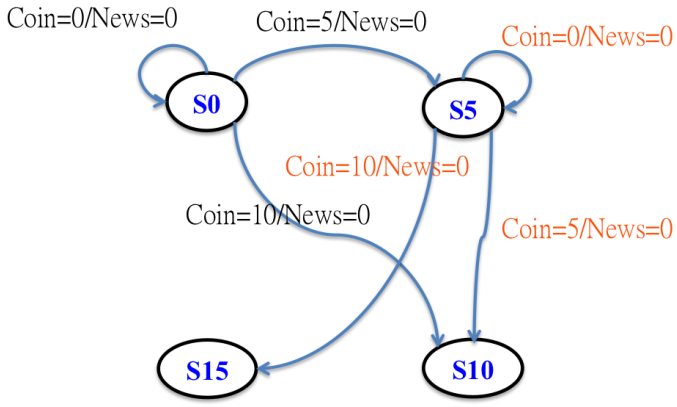
A=B
A<B
A>B

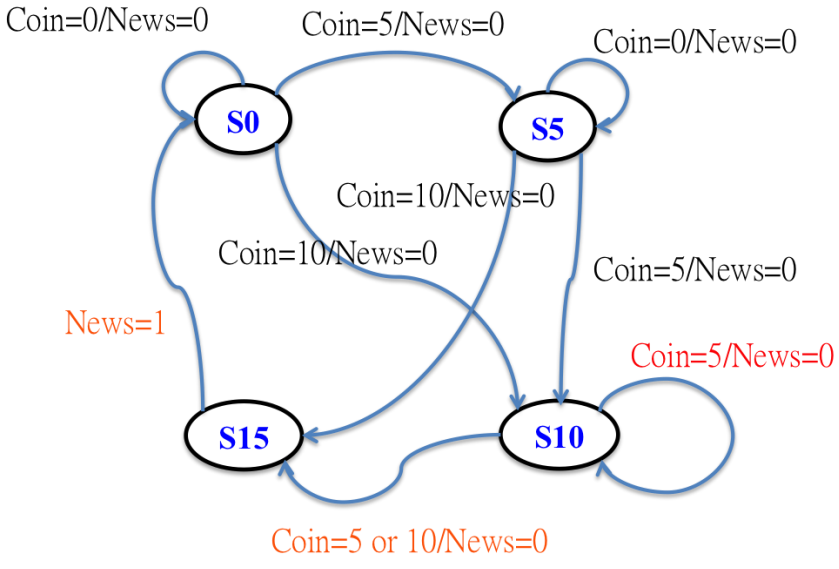
Valid combination:

(a) 5+5+5, (b) 5+5+10 (no return $), (c) 5+10,
(d) 10+5, (e) 10+10 (no return $)

S0=0 cent, S5=5 cent, S10=10 cent, S15=equal or over 15 cent







6.5.2 Newspaper Vending Machine

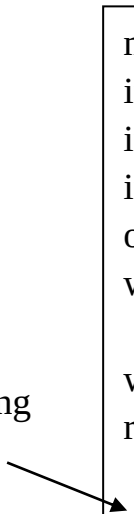
A simple digital circuit is to be designed for the coin acceptor of an electronic newspaper vending machine.

*Assume that the newspaper cost 15 cents.
*The coin acceptor takes only nickels and dimes.
*Valid combinations including order of coins are one nickel and one dime, three nickels, or one dime and one nickel. Two dimes are valid, but the acceptor does not return money.

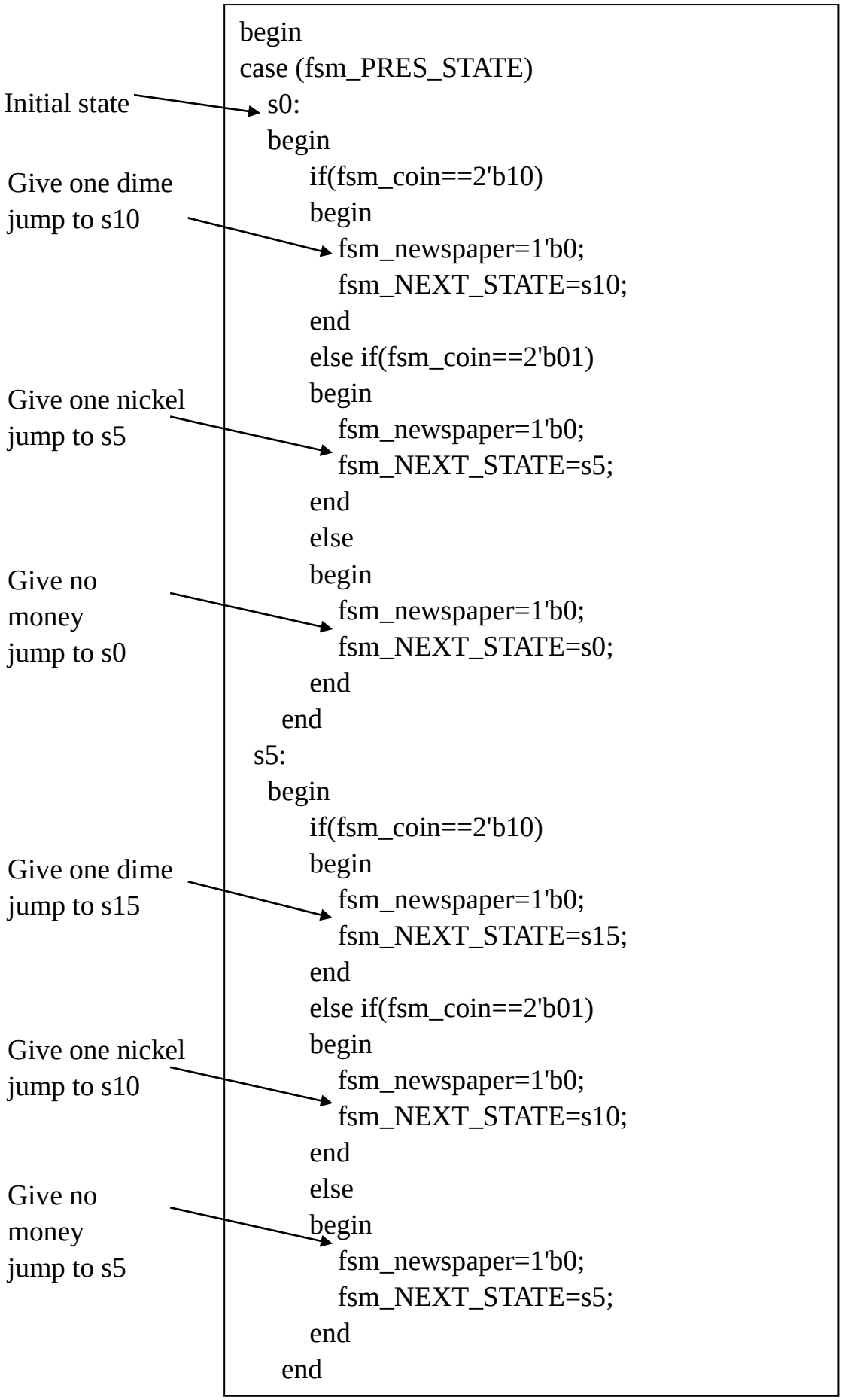
Verilog Code

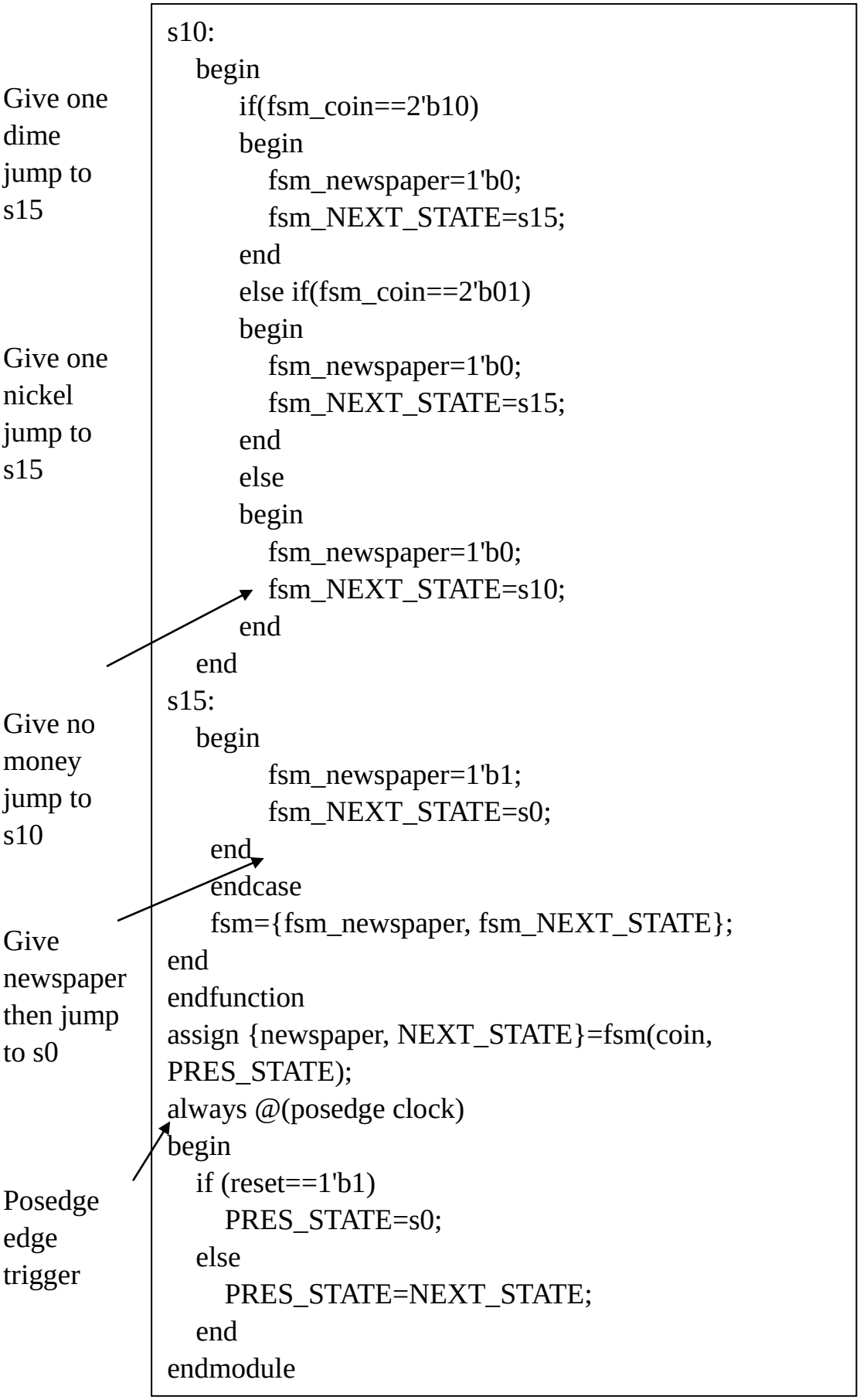
The Verilog RTL description for finite state machine is shown below.

State encoding



```
module vend(coin,clock,reset,newspaper);
input [1:0] coin;
input clock;
input reset;
output newspaper;
wire newspaper;

wire [1:0] NEXT_STATE;
reg [1:0] PRES_STATE;

parameter s0=2'b00;
parameter s5=2'b01;
parameter s10=2'b10;
parameter s15=2'b11;

function [2:0] fsm;
input [1:0] fsm_coin;
input [1:0] fsm_PRES_STATE;

reg fsm_newspaper;
reg [1:0] fsm_NEXT_STATE;
```

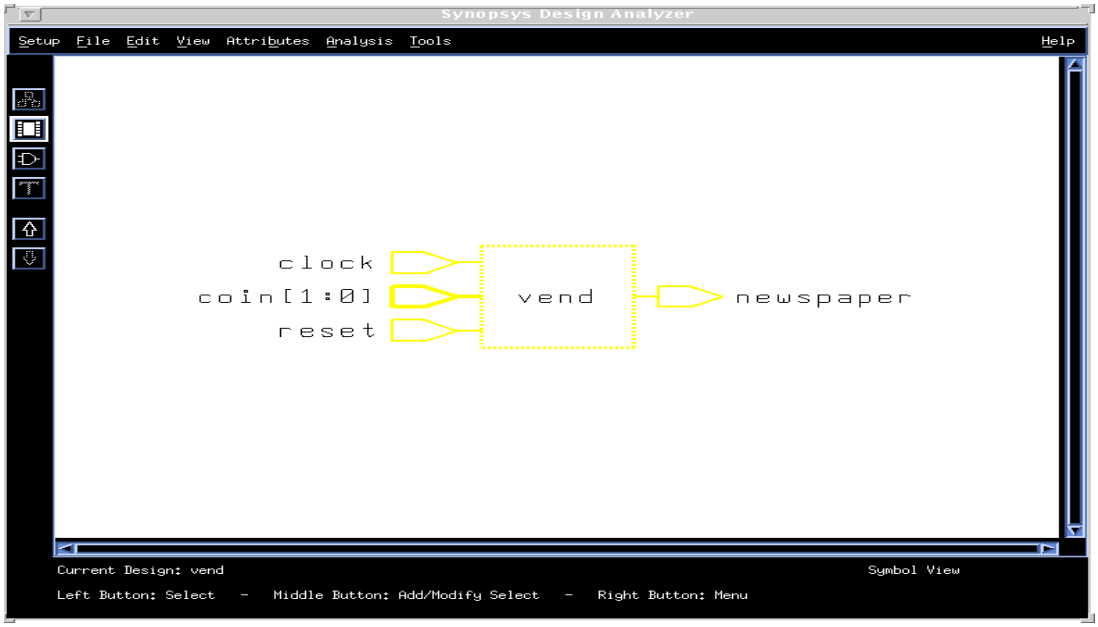

Initial state
Give one dime
jump to s10
Give one nickel
jump to s5
Give no
money
jump to s0
Give one dime
jump to s15
Give one nickel
jump to s10
Give no
money
jump to s5
begin
case (fsm_PRES_STATE)
s0:
begin
if(fsm_coin==2'b10)
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s10;
end
else if(fsm_coin==2'b01)
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s5;
end
else
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s0;
end
end
s5:
begin
if(fsm_coin==2'b10)
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s15;
end
else if(fsm_coin==2'b01)
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s10;
end
else
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s5;
end
end


Give one dime jump to s15
Give one nickel jump to s15
Give no money jump to s10
Give newspaper then jump to s0
Posedge edge trigger
s10:
begin
if(fsm_coin==2'b10)
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s15;
end
else if(fsm_coin==2'b01)
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s15;
end
else
begin
fsm_newspaper=1'b0;
fsm_NEXT_STATE=s10;
end
end
s15:
begin
fsm_newspaper=1'b1;
fsm_NEXT_STATE=s0;
end
endcase
fsm={fsm_newspaper, fsm_NEXT_STATE};
end
endfunction
assign {newspaper, NEXT_STATE}=fsm(coin, PRES_STATE);
always @(posedge clock)
begin
if (reset==1'b1)
PRES_STATE=s0;
else
PRES_STATE=NEXT_STATE;
end
endmodule

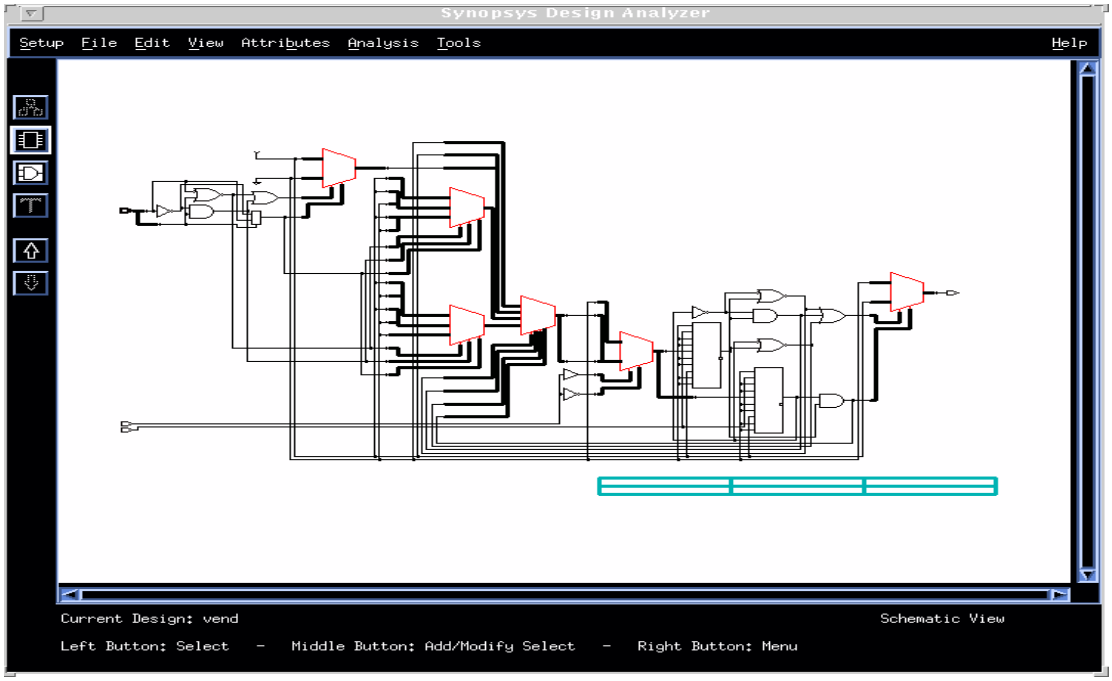
Synthesis Result

The logic synthesis tool produces gate-level description. The symbol and schematic for the gate-level circuit are shown below.

Symbol View



Schematic View



Test Stimulus Code

Stimulus is applied to the original RTL description to test all possible combination of coins. The same stimulus is applied to test the optimized gate-level netlist. Stimulus applied to both the RTL and gate level netlist is shown below.

```
module stimulus;
reg clock;
reg [1:0] coin;
reg reset;
wire newspaper;

vend vendy(coin, clock, reset, newspaper);
initial
begin
    $display("\t\tTIME       RESET NEWSPAPER\n");
    $monitor("%d %d %d",$time,reset,newspaper);
end
initial
begin
clock=0; coin=0; reset=1;
#50 reset=0;
@(negedge clock);
#80 coin=1;    #40 coin=0;
#80 coin=1;    #40 coin=0;
#80 coin=1;    #40 coin=0;
#80 coin=1;    #40 coin=0;
#80 coin=2;    #40 coin=0;
#80 coin=2;    #40 coin=0;
#80 coin=2;    #40 coin=0;
#180 coin=2;    #40 coin=0;
#180 coin=1;    #40 coin=0;
#80 $finish;
end
```

Test all possible combination

Generate periodic clock

```
always
begin
   #20 clock=~clock;
end
endmodule
```

Simulation result

Stimulation result is shown below. Thus, the gate level netlist is verified.

```
    LINIT begins
                TIME    RESET NEWSPAPER

                   0      1      |x
    LINIT done

    0 State changes on observable nets.

    Simulation stopped at the end of time 0.
Ready: sim
                  20      1       0
                  50      0       0
                 420      0       1
                 460      0       0
                 660      0       1
                 700      0       0
                 900      0       1
                 940      0       0
                1340      0       1
                1380      0       0

    245 State changes on observable nets.

    Simulation stopped at the end of time 1440.
Ready:
```