Lab 6

Cyclic Behavior

Learning objective of this lab

* Procedural assignment using cyclic behavior
* Pre-Lab
  1. Introduction

Cyclic Behaviors in verilog are used in Dataflow/RTL Models and Algorithm-Based Models. They can be used to model both combinational and sequential circuits. Today we will practice combinational circuits. Cyclic behaviors are abstract. In this mode of description, circuits are described using procedural assignment instruction (just as in C), showing relationship between input and output. So, the coder shouldn’t be worried about generating Boolean equation from truth table. The synthesizer generates Boolean equation based on the description.

* 1. Code Structure
     1. Operators

|  |  |
| --- | --- |
| **Function** | **Operator** |
| Less than | < |
| Greater than | > |
| Less than or Equal to | <= |
| Greater than or Equal to | >= |
| Equal to | == |
| Not Equal to | != |
| Equal to including ‘x’ and ‘z’ | === |
| Not Equal to including ‘x’ and ‘z’ | !== |

Table 6.1 (Relationship operators)

Relationship operators are used in if-else e.g if (a>=b). The difference between ‘==’ and ‘===’ is that ‘==’ will return ‘x’ if any of the operands contains an ‘x’; whereas ‘===’ compares ‘x’ and ‘z’ too, in the operands.

|  |  |
| --- | --- |
| **Function** | **Operator** |
| Add | + |
| Subtract | - |
| Multiply | \* |
| Divide (Truncates fractional part) | / |
| Remainder | % |

Table 6.2 (Arithematic operators)

* + 1. Instructions
       1. Cyclic Behavior

The keywords for a cyclic behavior are always @ (input). Just like initial behavior a code can have multiple cyclic behavior blocks. Contrary to initial block, which runs unconditionally, and, runs just once; cyclic behavior runs only when there is change in the variables written after @, and then, again starts waiting for change. All the inputs should be mentioned after @ for a combinational circuit. For example if a circuit has four inputs inp1, inp2, inp3 and inp4

always @(inp1 or inp2 or inp3 or inp4)

The instructions inside the always block are written using procedural assignment operator ‘=’. The variable on the left side of assignment operator should be of reg type. Register variables store information during simulation, but do not necessarily imply that the synthesized circuit will have hardware registers. If there are more than one instructions inside the block, they are enclosed by begin and end

For example the code for a Full adder using cyclic behavior is as follows.

module fulladder(result, x)

output reg [1:0] result; // 1 bit for sum and 1 for carry

input [2:0] x; //A fulladder has 3 inputs

always @ (x)

result=x[0] + x[1] + x[2];

endmodule

OR

module fulladder(sum,cout,a,b,cin)

output reg sum,cout; // 1 bit for sum and 1 for carry

input a,b,cin; //A fulladder has 3 inputs

always @ (a or b or cin)

{cout,sum}=a + b +cin;

endmodule

* + - 1. Conditional Assignment

Conditional assignment is implemented using if-else or case statements. If there are more than one statement inside an if or else condition it is enclosed by begin – end. Complete case is considered a single statement, but if inside case statement any condition contains more than one instructions, they are enclosed by begin – end.

For example implementation of 32-bit 4 to 1 mux using if-else is as follows. parameter makes the code more convenient for changing later and also makes code easier to understand.

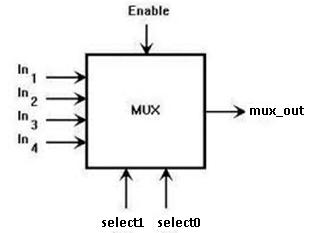


Figure 6.1 (4-1 Mux)

1. **4-1** Mux using if-else Statements

**module Mux\_4\_32(mux\_out, in3, in2, in1,in0, select,enable);**

**parameter dsize = 32;**

**output reg [dsize-1 : 0] mux\_out;**

**input [dsize-1 : 0] in3, in2, in1 , in0;**

**input [1 : 0] select;**

**input enable;**

**always @ (in3 or in2 or in1 or in0 or select or enable)**

**begin**

**if (enable==0)**

**mux\_out=32’bz;**

**else if (select == 0) mux\_out = in0;**

**else if (select == 1) mux\_out = in1 ;**

**else if (select == 2) mux\_out = in2 ;**

**else if (select == 3) mux\_out = in3 ; else mux\_out = 32'bx; //Don’t Care**

**end**

**endmodule**

The last condition “**else mux\_out = 32'bx;”** is important because it helps in reduction and without it, a latch is synthesized. The output should be specified for all possible combination of inputs. Synthesis tool, for the unspecified inputs, assumes retaining the previous value and hence a latch is synthesized.

1. 4-1 Mux using Case Statements

The code given below is an implementation of a 32-bit, 4-1 multiplexer using case.

**module Mux\_4\_32(mux\_out, in3, in2, in1,in0, select,enable);**

**parameter dsize = 32;**

**output reg [dsize-1 : 0] mux\_out;**

**input [dsize-1 : 0] in3, in2, in1 , in0;**

**input [1 : 0] select;**

**input enable;**

**always @ (in3 or in2 or in1 or in0 or select or enable)**

**begin**

**if (enable==1)**

**mux\_out=32’bz;**

**else**

**case (select)**

**2’b00: mux\_out = in0; //if select=00**

**2’b01: mux\_out = in1; //if select=01**

**2’b10: mux\_out = in2; //if select=10**

**2’b11: mux\_out = in3; //if select=11**

**default: mux\_out = 32'bx;//Any other value**

**// e.g. xx or zz**

**endcase**

**end**

**endmodule**

The circuit above can also be implemented using combination of continuous assignment and cyclic behavior.

**module Mux\_4\_32(mux\_out, in3, in2, in1,in0, select,enable);**

**parameter dsize = 32;**

**reg [dsize-1 : 0] any\_temp\_reg;**

**output [dsize-1 : 0] mux\_out;**

**input [dsize-1 : 0] in3, in2, in1 , in0;**

**input [1 : 0] select;**

**input enable;**

**assign mux\_out = enable ? any\_temp\_reg : 32'bz;**

**always @ (in3 or in2 or in1 or in0 or select)**

**case (select)**

**2’b00: any\_temp\_reg = in0;**

**2’b01: any\_temp\_reg = in1 ;**

**2’b10: any\_temp\_reg = in2;**

**2’b11: any\_temp\_reg = in3;**

**default: any\_temp\_reg = 32'bx;**

**endcase**

**end**

**endmodule**

* + - 1. Non-Blocking assignment operator

All the assignments written using <= execute in parallel so the order in which they are listed has no effect.

For example, if a=1, b=2, c=3

b=c

a=b will result in a=c=3

but

b<=c

a<=b will result in b=3, and a=2

1. **SR-Latch usig non-blocking assignment operator**

**module srlatch(input S,R,output reg Q,Qnot);**

**always @(S or R)**

**begin**

**Q<=~(S&Qnot);**

**Qnot<=~(R&Q);**

**end**

**endmodule**

* + - 1. Describing a Flipflop using posedge

A flipflop is described using cyclic behavior which is sensitive to edge of a signal (clock). Reset can be choosen as synchronous or asynchronous.

* + D-flipflop with Synchronous Reset (Synchronized with clock)

Reset will be checked at positive edge of clock

input clock,reset,D;

output reg Q;

always @ ( posedge clock)

begin

if(reset)

Q=0;

else

Q=D;

end

* + D-flipflop with Asynchronous Reset
    - Edge Triggered Reset

Reset is independent of clock.

input clock,reset,D;

output reg Q;

always @ ( posedge clock or negedge reset)

begin

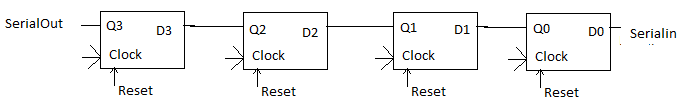
if(reset)

Q=0;

else

Q=D;

1. **4-bit Left Shift Register**



module shiftreg\_4bit(input serial\_in,

clock,reset,

output serialout);

reg [3:0]shiftreg;

always @(posedge clock or negedge reset)

begin

if(reset==0)

shiftreg<=0;//Non-Blocking Assignment operator

else

begin

shiftreg[0]<=serial\_in;

shiftreg[1]<=shiftreg[0];

shiftreg[2]<=shiftreg[1];

shiftreg[3]<=shiftreg[1];

end

end

endmodule

* In-Lab

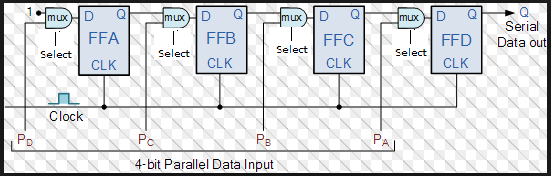
**Task(use Vectors)**

**Write and verify(through testbench simulation) Verilog HDL code for**

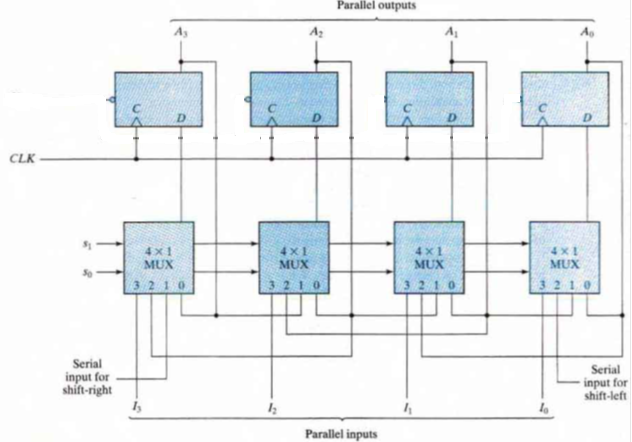
* **a BCD to 7 segment decoder.** For example for input 7 which is 0111, ouput abcdefg=1110000



* **following shift register**. Write two always behaviors. One for the flipflops and one for the muxes.

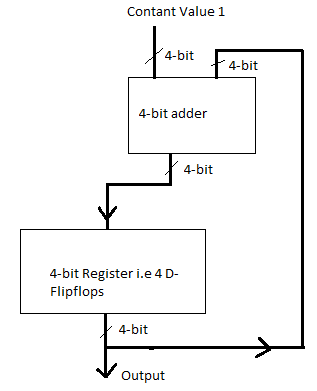


* **Universal shift register**. Waveform should display all inputs and contents of register for different select options. Write two always behaviors. One for the flipflops and one for the muxes.

****

**Post Lab**

* **4-bit counter**

****

**Submission details**

* **Your lab report, a .doc file, should contain properly commented Post-Lab task code, with Screenshots(of print preview) of Schematic and waveforms, and Critical Analysis.**
* **The report must have a title page in the pescribed format.**
* **Name the .doc file RegNo.docx; eg SP14-BCE-99.docx**
* **Sumbit on portal.**