Lab session-5

Learning Outcomes:

a) Designing a 32-bit register file for a MIPS processor,
b) Clock gating and integration of components.

Designing a 32-bit register file

Register File parameters

Width of each Registers- 32-bit;

No. of input ports- six;

- Three ports for addresses
- One port for data
- One for clock, one for *RegWrite*

no. of output ports= two;

Register file is comprised of following modules,

(a) **D-FF**,
(b) **Decoder** and,
(c) **Multiplexer**

Designing a 32-bit Register using D-FF

Functional Requirements are

1. Control input C
2. Data input D

generate

genvar i;

generate

for$(i = 0;\ i < N;\ i = i + 1)$

begin DFF $D1\,(\cdots)$;

end

endgenerate

Pseudo-code for the register

-Parallel in parallel out type

input clk, *clear*;

input$[31:0]$ *pi*;

output reg $[31:0]$*po*;

always @ (*posedge clk*)

begin

if (*clr*) *po* $<=$ $32'h0000$;

else po $<=$ *pi*;

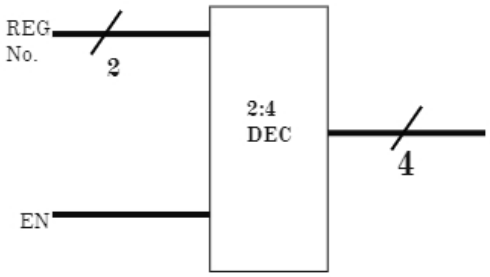
end

Design of Decoder

A *Decoder* should

(a) *select registers* as per the input register number

(b) be enabled (*en*)



Pseudo-code for decoder

always @(*en*, *in*)
begin if (*en*)*out* = 8′*b*0;
else case(*in*)
begin
2′*b*00 : *out*[0] = 1′*b*1;
2′*b*01 : *out*[1] = 1′*b*1;
2′*b*10 : *out*[2] = 1′*b*1;
2′*b*11 : *out*[3] = 1′*b*1;
default : *out* =
endcase
end

Design of the Multiplexor

Pseudo-code for Mux

$input\,[1:0]\,reg_{no}$

$input\,[31:0]\,a_0, a_1, \ldots;$

$output[31:0]\,out_{mux};$

$always\,@\,(*)\,begin$

$case(reg_{no})$

$2'b00 : out_{mux} = a_0;$

$2'b01 : out_{mux} = a_1;$

$2'b10 : out_{mux} = a_2;$

$2'b11 : out_{mux} = a_3;$

$default : out$

$endcase$

end

Remark:

- *case* statements should have a *default* condition.
- *if* statements should have a corresponding *else*.

Design of Clock gating Cell

Definition of ICG

ICG, otherwise known as **I**ntegrated **C**lock **G**ating cell is placed to **have power saving**.

Pseudo-Code of ICG

$module(\cdots)$

$input\ reg-clk, reg-write, decoder-out;$

$output\ gated-clk;$

$and(gated-clk, reg-clk, reg-write, decoder-out)$

$endmodule$

Exercises

Modularity increases $re - usability$ and reduces the time for bug-fixes.

Integrate the following modules:

(A) *Decoder* [5 : 32]

(B) Mux [32:1]

(C) Registers 32-bit [32]

(D) ICG