Assgn2_Grp_27

Assignment 2 of Verilog
Group 27
Yash Sirvi (21CS10083)
Sanskar Mittal (21CS10057)

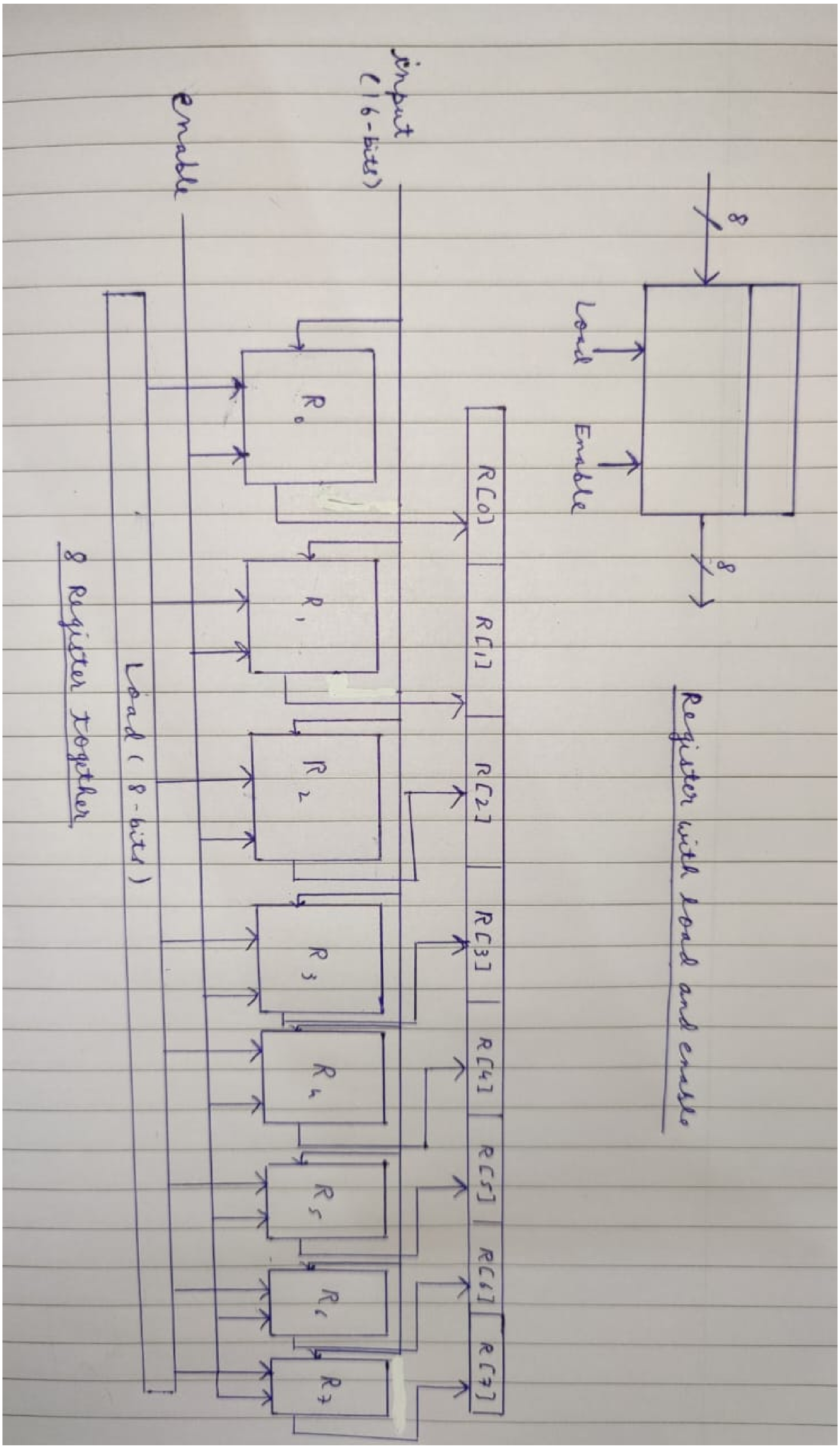
Q1 Register control

load_enable_register module

Input:

- `load` and `enable` lines as control signals for the register
- `data` is a 16 bit data line for data transfer into a register
  Output:
- `out` for getting data out of the register
  Other registers:
- `r` register is a 16 bit register holding the value of the current register module
  Behavior
- If `load` is set to 1, we load the data inside `data` into `r`.
- If `enable` is set to 1, we output the data stored in `r` to the `out` line otherwise, we set the output to high impedence `z`
- These conditions enables our module to act as a register with load enable functionality where it can take data from a bus and also supply data to a bus given proper control signal

Schematic



main_register module

- This our top level module that uses the load_enable_register inside it
  Input:
- Three bit `source` and `dest` for selecting the correct register
- `move` and `in` control signals
- `inp_d` is the input data in case of `in` control signal
- `clk` is clock signal for driving the entire module
  Output
- 16 bit `out` register for getting the value of the selected destination register
  Other Registers
- `R` stores 8 register of size 16 bits
- `enable` to set control signal for the register
- `load` is a 8 bit value where each bit corresponds to the load control signal for each Register
- `inp_i` is a 16 bit register as the input value to the register module
  Behavior
  We instantiate 8 instances of register load_enable_register module
  On every clock's positive edge, we check if the load and enable are both 1 or not. If yes then we print an error as we cannot load and store a register into the bus at the same time.
  Otherwise, we check for if move is on or not, select the source and destination register and move the values form source to destination.
  If in is 1 then we load the value given in the inp_d into the destination register.

Top Level module

Schematic

Q2. GCD

Input:

- x and y are two 8 bit inputs
- clk is clock signal that will drive the entire circuit

Output:

- out: this is the final calculated gcd output

Other Registers

- `temp1` and `temp2` are two registers which are initialised with x and y. All the calculations are done on these temporary registers
- `calculating`: this is used to infer if a new value of x and y is given as input during each clock cycle

Behavior

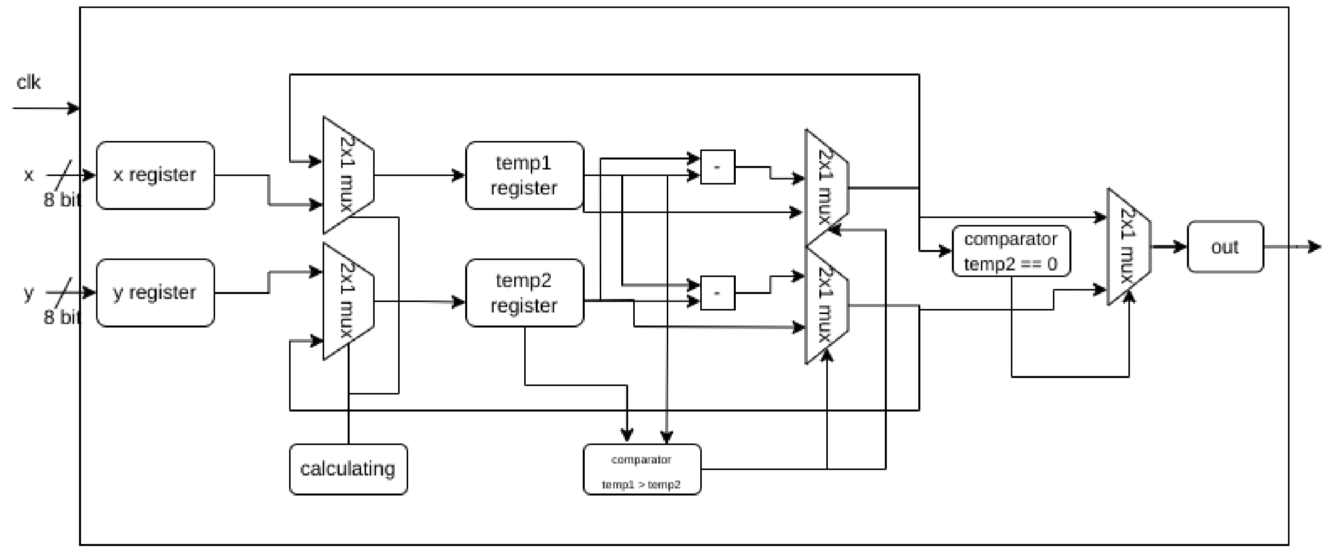
- All the operations take place during the positive edge of the cycle
- Whenever x or y changes, `calculating` goes to 0 indicating that `temp1` and `temp2` needs to be initialised. After the initialisation, `calculating` is set to 1 and the actual calculation of gcd startes from next clock cycle.
- We use euclidian algorithm for calculating gcd. `temp1 = temp1 - temp2` if `temp1 > temp2` and similarity `temp2 = temp2 - temp1` if `temp2 > temp1`. We continuously do this until one of `temp1` or `temp2` becomes zero and then return the other register's value as the final result.

Testbench

- We use `always #1 clk = ~clk` to change the clock's signal every 1 unit of timescale
- We use `$display` to print the answers of our module

- We wait for 100 units of time before returning the final output of gcd

Schematic


clk
x
8 bit
y
8 bit
x register
y register
2x1 mux
2x1 mux
temp1 register
temp2 register
-
-
2x1 mux
2x1 mux
comparator temp2 == 0
2x1 mux
out
calculating
comparator
temp1 > temp2