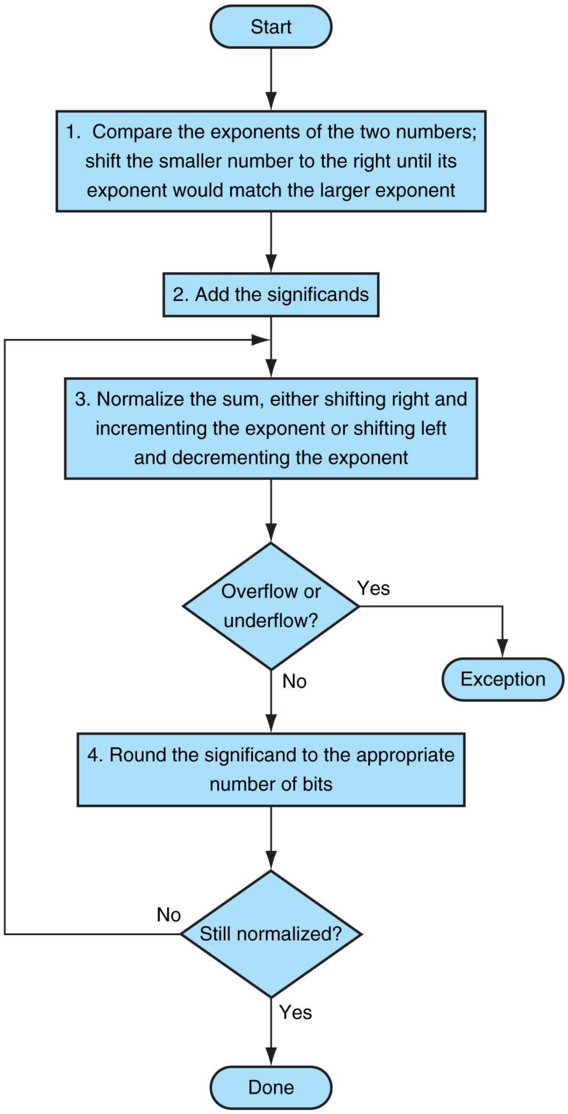
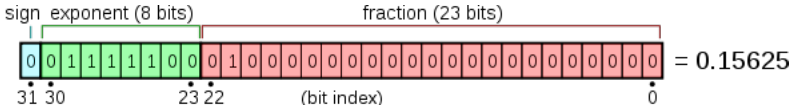
ECE 410/510
Introduction to SystemVerilog
Summer 2011
Homework 1

1. Use SystemVerilog to model and test a simple floating point adder. The adder should accept two inputs in IEEE 754 floating point format (single precision) and produce on its output the single precision sum of the numbers in IEEE 754 floating point format. You do not need to implement the IEEE 754 floating point rounding methods (using round, guard, and sticky bits); you can round to zero. You do not need to handle denormalized numbers.

   An algorithm for floating point addition is provided below.


Start
1. Compare the exponents of the two numbers; shift the smaller number to the right until its exponent would match the larger exponent
2. Add the significands
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow or underflow?
Yes
Exception
No
4. Round the significand to the appropriate number of bits
No
Still normalized?
Yes
Done

Single precision IEEE 754 floating point format is depicted below



$$= (-1)^{sign}(1.b_{-1}b_{-2}...b_{-23})_2 \times 2^{e-127}$$

Make use of SystemVerilog constructs and idioms where applicable. For example:

- Use a package for definitions that you need to share between the module defining the floating pointer adder and any model that might use the floating pointer adder (e.g. a higher level module or a testbench).

- Use a typedef to create a type type for IEEE 754 floating point numbers

- Use parameters and local parameters where/if appropriate

SystemVerilog provides a **`shortreal`** data type that may be useful in your testbench for generating test data. You can convert from that to an arbitrary 32-bit value with **`$shortrealtobits()`** and back with **`$bitstoshortreal()`**.

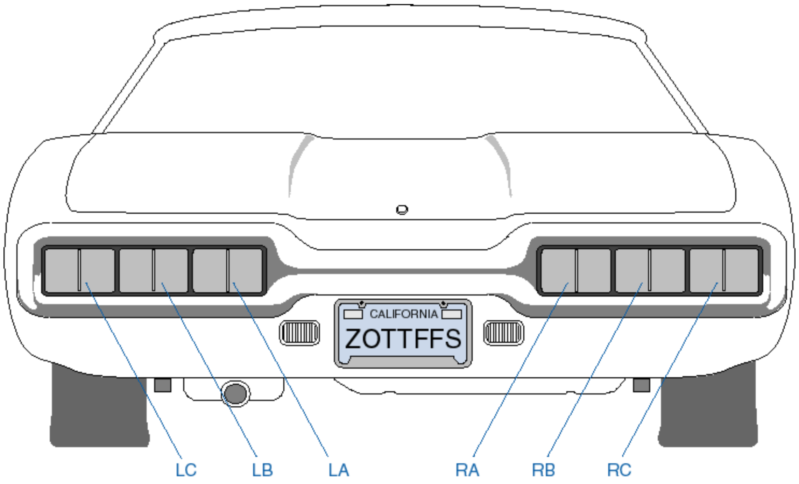
2. Download the T-bird tail light finite state machine implementation (model and testbench) from the course web and convert it to SystemVerilog, using appropriate SystemVerilog constructs, including but not limited to:

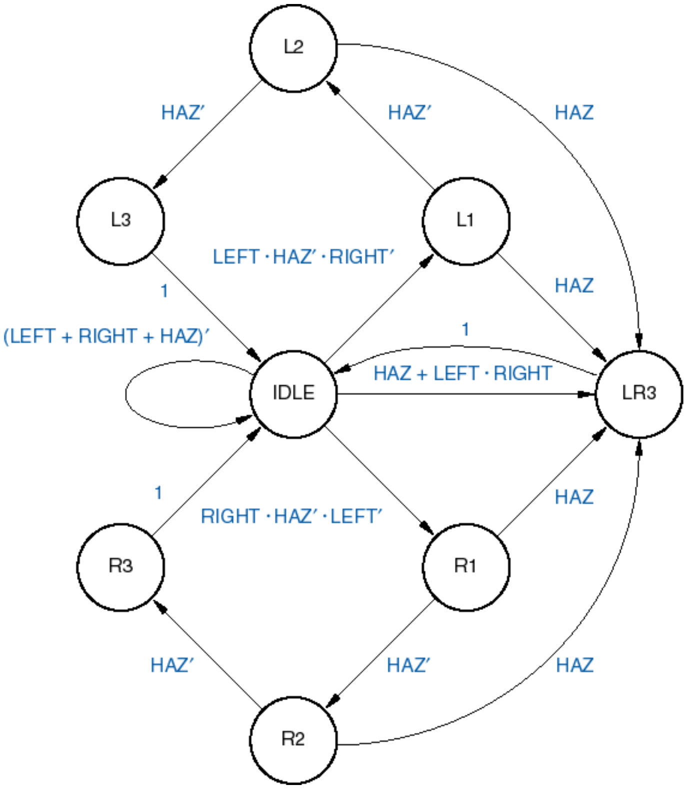
   - Use enumerated types for the states

   - Use time value suffixes for time values and assume a complete (uninterrupted) turning sequence should take two seconds

   - User parameters and local parameters where/if appropriate

   - In the testbench print the name of the state the FSM is in at every state change


CALIFORNIA
ZOTTFFS
LC
LB
LA
RA
RB
RC


L2
HAZ′
HAZ′
HAZ
L3
L1
LEFT · HAZ′ · RIGHT′
HAZ
1
(LEFT + RIGHT + HAZ)′
1
HAZ + LEFT · RIGHT
IDLE
LR3
1
RIGHT · HAZ′ · LEFT′
HAZ
R3
R1
HAZ′
HAZ′
HAZ
R2