Advanced Logic Design – Lab Number 4

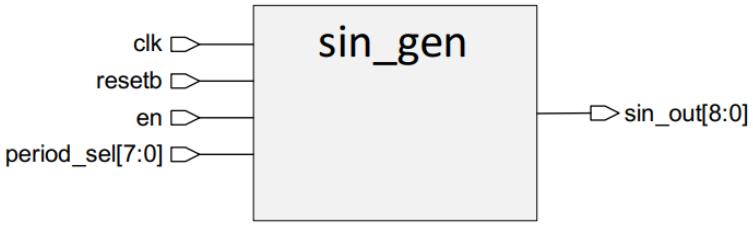
1. **(10 points)** Explain the usage of “disable fork” – why we need it? give an example.

2. **(10 points)** Consider two pipeline multipliers of 6 bits. Multiplier *A* is based on regular add-shift algorithm, while multiplier *B* is based on the very basic booth’s algorithm. If multiplicand (*M*) value is 6’b111111 and multiplier (*Q*) value is 6’b010101: How many cycles it will take for each one of the multipliers to finish the calculation? Explain.

3. **(10 points)** Draw a gate level diagram of the following SystemVerilog code.

```
always_ff @ (posedge clk or negedge rstb)
    if (~rstb)
        vec[3:0] <= 4'b0;
    else
        vec[3:0] <= {(vec[1] ^ din), vec[3:1]}
```

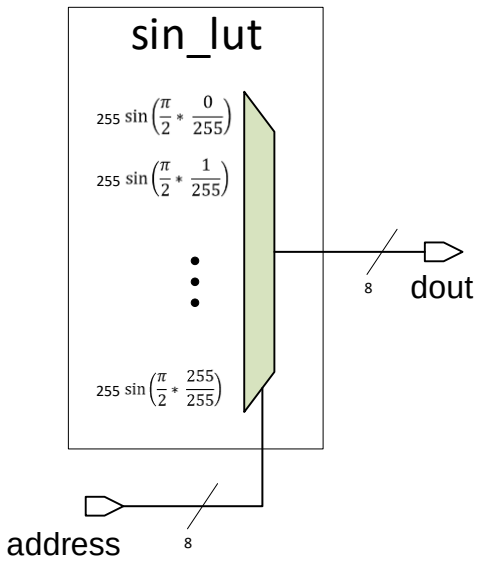
4. Design a block that generates a sinus wave with a tunable frequency.



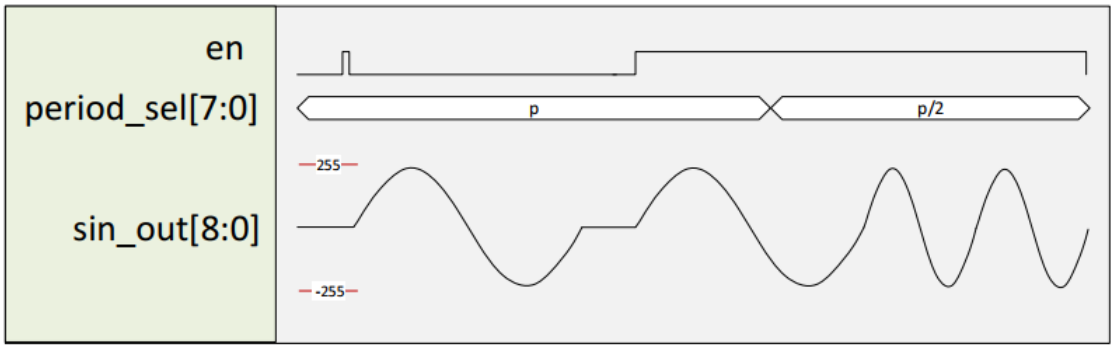
Port	Direction	Width	Description
clk	In	1	512Mhz clock
resetb	In	1	Asynchronous active low reset
en	In	1	Enable the sinus generation (active high)
period_sel	In	8	Define the sinus frequency (see the detailed below)
sin_out	Out	9	Two's complement value that represents a fractional value between -1 to +1

Operation Description

Given a LUT (look up table) module in SystemVerilog (sin_lut.sv) that contains 256 unsigned 8-bit samples of the first quarter of sinus.



Drive the sinus wave in 'sample & hold' method. While *period_sel*=0 each sample is hold for a single clock cycle and this is the highest sinus frequency that can be achieved. The actual sinus frequency in this case is 512M/256*4 = 500KHz. While *period_sel*=1 each sample is hold for 2 clock cycles, therefore the sinus frequency will be 2 times slower, and so on.
The sinus wave will start while the enable sign *en* is high. If *en* is de-asserted to 0, the current sinus cycle should be finished and then the output stays on 0 until the new enable.
Similarly, if *period_sel* is changed in the middle of a sinus cycle, it will not take effect immediately, but at the beginning of the next cycle. (See wave diagram below).
Note: All inputs/output are synchronized with the clk.



Implementation Guide

a) The design should be based on an FSM.

b) Draw a micro architecture diagram of your solution. Show the interaction between the sub-blocks to each other and between subblocks to top interface. (high level -deep details are not required here)

c) Design the FSM
- Define its inputs and outputs.
- Draw the state diagram.
- Write a SystemVerilog code.

d) Draw a diagram of your full design.
No need to draw the FSM implementation, the state diagram is enough.

e) Write the full SystemVerilog code according to your micro architecture.

f) Write a simple testbench to simulate your design.

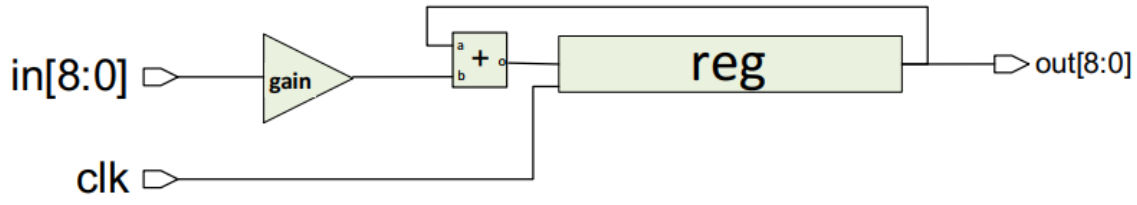
Delivery

I. **(10 points)** Micro architecture diagram
II. **(10 points)** FSM state diagram
III. **(10 points)** Full design diagram
IV. **(30 points)** The sin_gen.sv
V. **(5 points)** The sin_gen_tb.sv
VI. **(5 points)** Snapshot of the sinus wave changing frequency on the fly with the following values: en=1; period_sel=0; period_sel=1; period_sel=0;

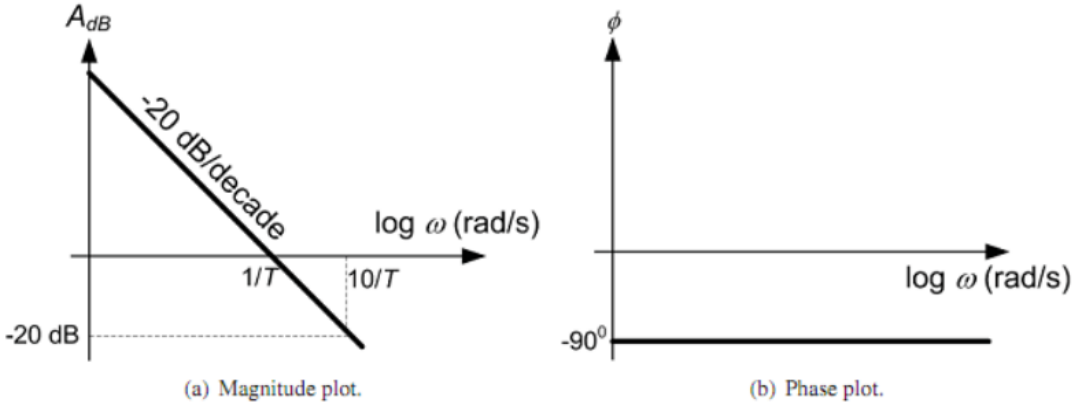
Bonus

Given an SV module of a digital integrator (integrator.sv). It is simply a register that gets to its input the result of the addition of its own value and the input value, divided by some factor. (See diagram below and sv code)
Instantiate the integrator in your top-level module. Route your sinus wave to the output pin through the integrator.

a) Set a maximal sinus frequency and compare the sinus wave in integrator input and output.
b) Set period_sel to 8'hf8 and compare again. What do you see now? Explain the differences in amplitude and phase. (Integrator transfer function graph is attached)



Digital integrator



Integrator transfer function graph