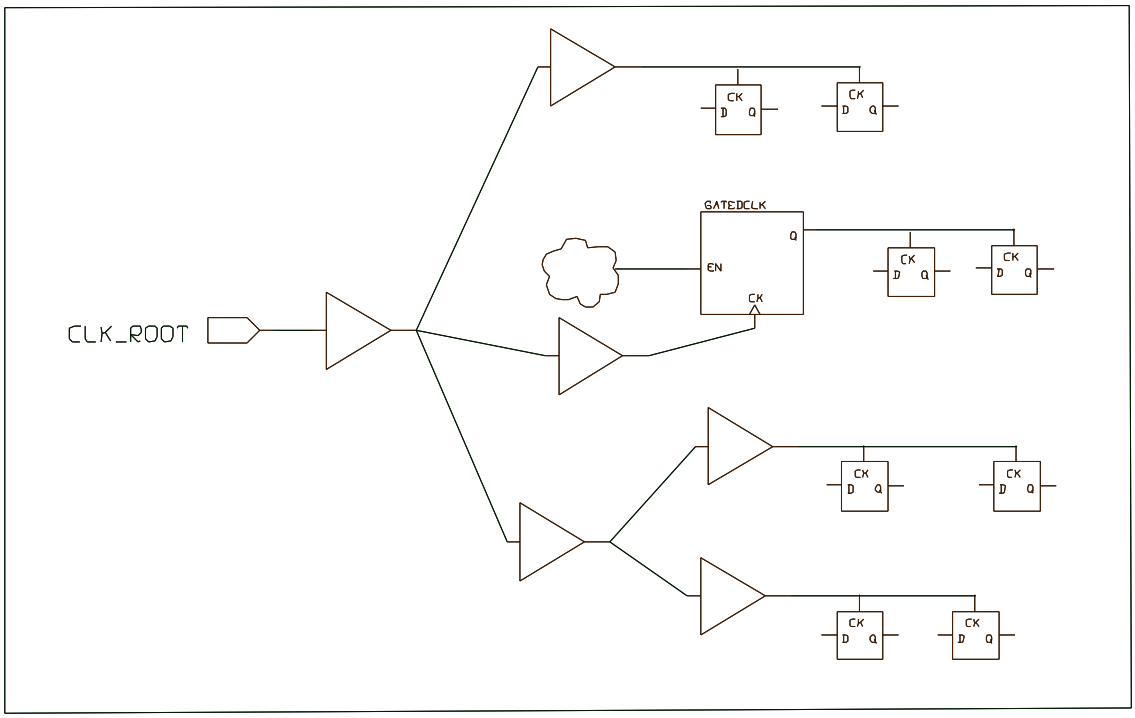
Clock Efficiency flow

What is Clock Efficiency

Each flop in an ASIC chip has a clock input. Usually this clock is a leaf of a clock tree (or one of the trees).
On edge (+ or -), this flop samples the data input "D" and transfers the new value to output "Q". If "D" causes a change in the "Q" it is a useful clock cycle. If clock pulses, but "Q" doesn't change it is wasted cycle. If "D" changes and clock doesn't pulse it is a dropped change.



Efficiency is defined the ratio between "useful" clocks and total number of clocks reaching the flop.
Most new designs have elaborate clock gating. If the clock tree is disabled early or in many branches, we save power.
Clock-tree power consumption is relatively significant, and to save battery, It is desirable to increase the ratio.
The way to do it, is rewrite RTL, so more flops will have enables.

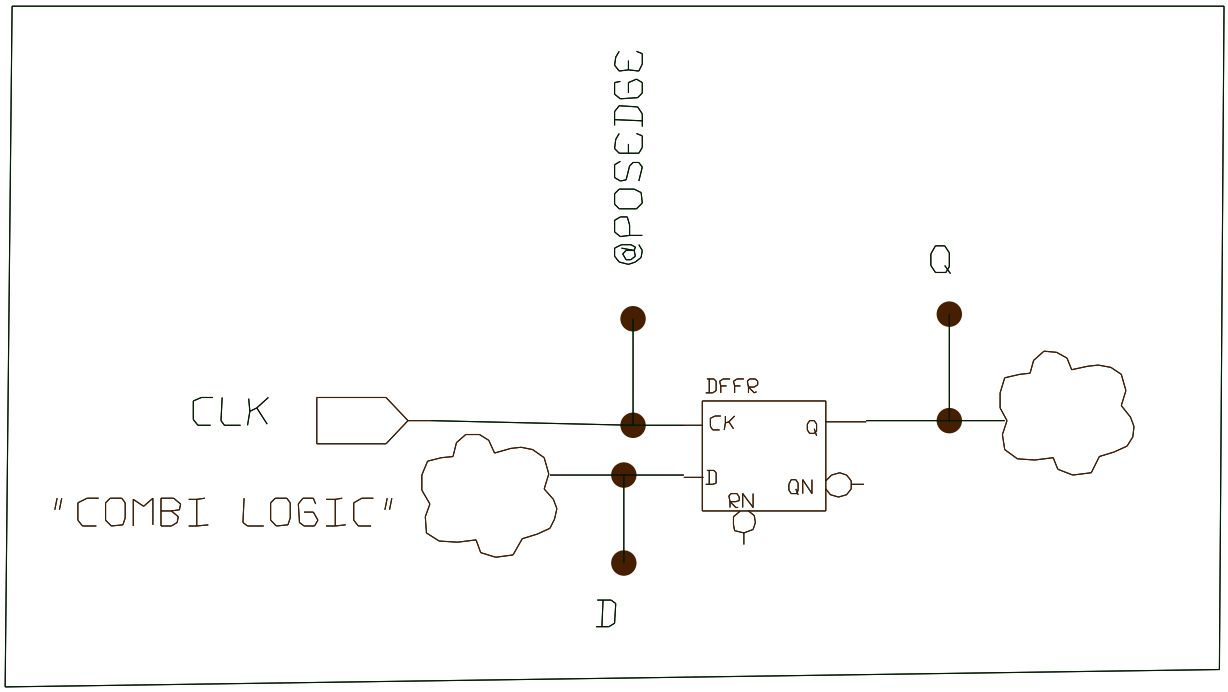
Also considering the number of "D" changes, when there is no clock. It means we waste energy toggling the "D" without using it.

The game : saving power.

For washing machine chips, this is not a big deal, as the red light in front of the machine consumes more power than all electronics. But for batterie powered devices or energy harvested from environment this could be a big deal.

How

There are several ways to get info on wasted clock cycles, It can be done on gate-level, where for every flop, we assign python monitor (generated by script).
This monitor catches the edge of the clock, peeks at "D" and compares to "Q". (Clock going up in simulation, precedes the change of Q). Note that to save power, CLK can be actually a gated clock. So environment counts the number of clock edges against difference of "D" and "Q". Ideally these two counts should be equal. In practice 50% is ok, because 100% is not conveying any new information. In practice, we can try to rewrite RTL to increase the ratio.



For rtl, script described below creates class that monitors one module (or deep into one module), gathers the counts and reports in the end. The script collects all "always" blocks with Clocks qualified with "pos or neg edges" and all left side variables of "Qflop <= ". We also count the number of times "D" toggles and there is no clock.

The result is written into Python class that counts these scenarios.
The whole flow is supported by Python-Driven-Verification.

Third option, is to use same python class as before, but run it on VCD file from simulation using vcd_python3 executable.

Preparation

pyver.py rtl/my_module.v -do clock_efficiency_rtl
(pyver.py and clock_efficiency_rtl.py are in gitHub repository vlsistuff/verpy/pybin)
this should create a file my_module_efficience.py

Detailed scripts "run" is in vlsistuff/clock_efficiency/examples/

Simulation (requires pythnon-driven-verification from the same git)

xrun/vcs/iverilog -f **YourFileList** .....

at the end of the test, report is printed out into pymon.log0

how it really looks

RTL code:

```
always @(posedge clk or negedge reset_n) begin
   if (~reset_n)
     rd_trans <= 1'b0;
   else if (rd_start | resume_rd_start)
     rd_trans <= 1'b1;
   else if (rd_end)
     rd_trans <= 1'b0;
end
```

Using pyver.py system with clock_efficiency.py driver, Will create the following code:

```
def run_posedge_clk(self):
     self.Touched = []
     self.Peeked = {}
     if self.evaluate_valid(['||', ['!', 'reset_n'], ['&&', ['|', 'rd_start', 'resume_rd_start'], 'reset_n'],
         ['&&', 'rd_end', ['&&', ['!', ['|', 'rd_start', 'resume_rd_start']], 'reset_n']]]):
         if self.changed("rd_trans"):
             self.increment_work("rd_trans")
         else:
             self.increment_waste("rd_trans")
```

Not shown is the class definitions and other counters. Also some optimization can reduce the complexity of this expression.

Running any simulation with this class imported and constructed, will yield report something like :

```
@54164: info: mymodule      rx_address      used=   16   wasted=   0   width=31
@54164: info: mymodule      rx_error        used=    1   wasted=  15   width=2
@54164: info: mymodule      rx_psram        used=    2   wasted=  14   width=1
@54164: info: mymodule      rx_loopback     used=    1   wasted=  31   width=1
@54164: info: mymodule      rd_trans        used=    1   wasted=  15   width=1
~
```

another example of log file, with DROPPED enabled. Notice, negative dropped values, it happens when the new data coming from literal values.

```
@170850: info: ONFINISH of clock efficiency in uartx2
@170850: info: uartx2   rxd1          used=  557   wasted=  170287   dropped=       0   width=1
@170850: info: uartx2   rxd_raw       used=  557   wasted=  170287   dropped=       0   width=1
@170850: info: uartx2   rxd_filt      used=    0   wasted=       0   dropped=     557   width=1
@170850: info: uartx2   last_rxd      used=  557   wasted=  170287   dropped=       0   width=1
@170850: info: uartx2   rxsr          used= 1100   wasted=     100   dropped=     379   width=11
@170850: info: uartx2   rx_noise      used=    1   wasted=    1099   dropped=     200   width=1
@170850: info: uartx2   rx_bufa       used=   99   wasted=       1   dropped=    1002   width=9
@170850: info: uartx2   ufb_valid     used=  199   wasted=       0   dropped=    -198   width=1
```

Width of the flop registers is important here, because it multplies the effect. The so first line is very good, while others are not and need to be examined for opprtunity to improve power consumption. The sum of "used" and "wasted" counters is the number the clock of these flops was toggled.

With this new info, we can scan the RTL and try to improve ratios of the worst offenders.

work in progress

The question is how efficient is this flow to improve power consumption. I am doing few trials and results are updated in **vlsistuff/clock_efficiency** . This doc is in **vlsistuff/docs**