VARIABLE FREQUENCY CLOCKING HARDWARE
Many complex digital systems have components clocked at different frequencies.

- Reason 1: to reduce power dissipation
  - The active component of power is proportional to the clock frequency
  - If a module’s clock frequency can be reduced while maintaining acceptable performance, its frequency can be reduced to reduce power dissipation

- Reason 2: Because a sub-module requires a specific clock frequency that is different than the main system’s frequency.
  - For example, the DDR4-3200 synchronous DRAM memory interface has an I/O bus that operates at 1.60 GHz and so the module certainly requires a 1.60 GHz clock
Multi-rate Clocking Hardware

1) Build slower divided clocks with FFs
   - Some FFs are clocked by the real $\textit{clock}$ signal, others are clocked by a delayed slower $\textit{clock}_{\text{freq-half}}$ signal coming from a frequency divider. Significant clock skew $\rightarrow$ potential for dead chip 😞
   - Could risk your job security (moderate exaggeration)

2) Use multi-frequency clocks
   - Requires an independent and complete clock tree for each frequency and possibly an independent phased-locked loop (PLL)
   - May save significant power in main processing circuits
   - Each PLL uses significant power
3) Pseudo-multi-rate: Clock all logic with highest-rate clock
   – Utilize simple counters that load registers or route signals on only certain clock edges (for example, every fourth clock edge for $freq/4$).
   – Counters must be reset simultaneously and the reset signal must meet timing requirements at the highest frequency
   – Definitely the simplest and most robust
   – Design in only this way in this class
3) Pseudo-multi-rate: Clock all logic with highest-rate clock
   - Possible issue if there are a very large number of FFs requiring the same enable signal
     1. delay of the fanout tree reduces available cycle time
     2. the enable signal could be modestly pipelined
   - Effectiveness
     + Logical operation—same as if frequency was reduced
     + Power reduction
       of logic—same as if frequency was reduced
   - Power reduction
     of clock signal—
     not at all

\[\text{Enable} = 1\]
\[\text{every fourth cycle is equivalent to using } freq/4\]
Multi-rate Clocking Hardware

- Example 1a to imitate a clock frequency of \( \text{freq}/4 \)

```verilog
reg [1:0] count, count_c; // two bits counts 00, 01, 10, 11, 00, ...

always @(*) begin
    // defaults
    count_c = count + 2'b01; // just let the counter wrap 2'b11 \rightarrow 2'b00
end

// en_freq4 will be high 1/4 of the time
reg en_freq4;
always @(*) begin
    if (count == 2'b00) begin
        en_freq4 = 1'b1;
    end
    else begin
        en_freq4 = 1'b0;
    end
end

// breaking a guideline with "if" here
always @(posedge clk) begin
    count <= #1 count_c;
    if (en_freq4 == 1'b1) begin
        Q <= #1 D;
        state <= #1 state_c;
    end
end

PLL

en_freq4 == 1 every fourth cycle is equivalent to using \( \text{freq}/4 \)
```
Multi-rate Clocking Hardware

- Example 1b to imitate a clock frequency of \( \frac{freq}{4} \)

```verilog
reg [1:0] count, count_c;
// two bits counts 00, 01, 10, 11, 00, ...
always @(\*) begin
  // defaults
  count_c = count + 2'b01;
  // just let the counter wrap 2'b11 \( \rightarrow \) 2'b00
end

wire en_freq4;
// use a wire in this example
assign en_freq4 = (count == 2'b00); // code is compact but slightly less clear

// breaking a guideline with “if” here
always @(posedge clk) begin
  count <= #1 count_c;
  if (en_freq4 == 1'b1) begin
    Q <= #1 D;
  end
end
```

PLL

\( en_{freq4} = 1 \)
every fourth cycle is equivalent to using \( freq/4 \)

\( D \) \hspace{2cm} Q
Multi-rate Clocking Hardware

- Example 2 to imitate a clock toggling at 1 Hz, with 500 MHz clock

```verilog
define reg [28:0] count, count_c;
    // 29 bits counts up to 536 million
    define reg en_increment;
    // I use a reg in this example

always @(*) begin
    // defaults
    count_c = count + 29'0000_0001;  // a flip-flop register
    en_increment = 1'b0;             // a combinational logic signal

    if (count == 29'd499_999_999) begin
        count_c = 29'h0000_0000;       // wrap counter back to zero
        en_increment = 1'b1;           // pulse FF enable signal high
    end
end

always @(posedge clk) begin
    count <= #1 count_c;
    if (en_increment == 1'b1) begin
        Q <= #1 D;
    end
end
```

PLL

en_increment

D

Q