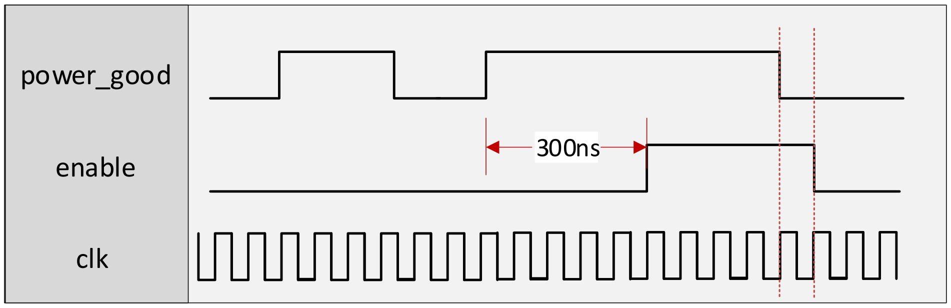
3. Implement the *power_on* module according to the spec below:

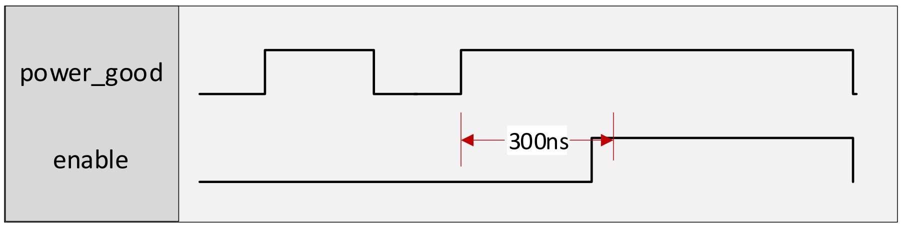
Port	Direction	Width	Description
resetb	In	1	Asynchronous active low reset
clk	In	1	Clock at 100Mhz
power_good	In	1	Power good indication
enable	Out	1	Enable signal. The *enable* should be asserted 300ns after *power_good* signal is asserted and stable



The Implementation should include:

a) **(5 points)** Block Diagram of the power_on module
b) **(10 points)** SystemVerilog code (**power_on.sv**)
c) **(15 points)** Testbench (**power_on_tb.sv**) that
- Print a message with the delay between the 2 signals
- Print error message if the *enable* rise less than 300ns after the *power_good* rising - based this checker on **"fork-join"** construct.

Fail



4. **(50 points)** Binary GCD Algorithm

Implement the Euclidean algorithm “Greatest Common Divisor“ (GCD) according to the following spec:

Pin name	**Direction**	**Description**
clk	In	100MHz clock
resetb	In	Asynchronous active low reset
u[7:0]	In	Unsigned, greater than 0 number
v[7:0]	In	Unsigned, greater than 0 number
ld	In	1 cycle pulse which tells that new u and v are ready and stable Assumption: time between 2 load pulses is long enough. New load will come only after previous calculation was finished.
res[7:0]	out	gcd(u,v) - Great Common Divisor of u and v. for example: if u = 36 and v = 24 res = 12 (after certain amount of cycles. there is no constraint on calculation speed) During the GCD calculation res = 0.
done	out	Indicate that the result at res[7:0] is valid (active high until the next load operation)

The GCD algorithm:

While (u ≠v) :

a) If both, u and v, are even: gcd(u,v) = 2*gcd(u/2, v/2)

b) If only u is even: gcd(u,v) = gcd(u/2, v)
   If only v is even: gcd(u,v) = gcd(u, v/2)

c) If both, u and v, are odd:
   - if (u>v) gcd(u,v) = gcd(u-v,v)
   - if (v>u) gcd(u,v) = gcd(u, v-u)

The GCD result is $\boldsymbol{u} * \mathbf{2}^{\boldsymbol{n}}$
while n is the number of times that both u and v were even (case a.)

The Implementation should include:

a) **(10 points)** Block Diagram of the GCD module
b) **(30 points)** SystemVerilog code (**gcd.sv**)
c) **(10 points)** A testbench (**gcd_tb.sv**) with a random test that is repeated 100 times. The test should display a message like: GCD (24,36) = 12 and response with error message if needed. For the golden model, implement a function that calculates the expected results using the regular Euclidean algorithm.