System-Level Simulation of a SPAD-Based Time-of-Flight Sensor in SystemVerilog

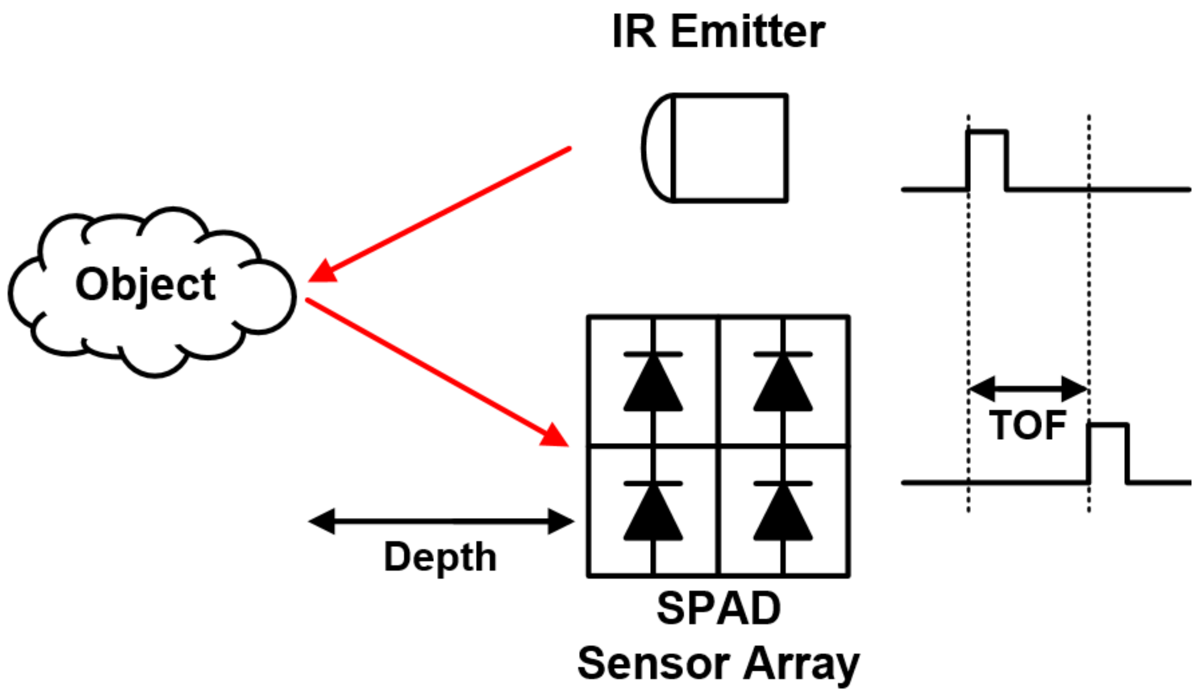
Seungah Park[1], Hyeongseok Seo[2], Canxing Piao[2], Jaemin Park[3], Jaehyuk Choi[1,2], Jung-Hoon Chun[1,2]

[1]Sungkyunkwan University, Suwon, Korea
[2]SolidVue Inc., Seongnam, Korea
[3]Scientific Analog Inc., Seoul, Korea

Contents

Direct Time-of-Flight Method

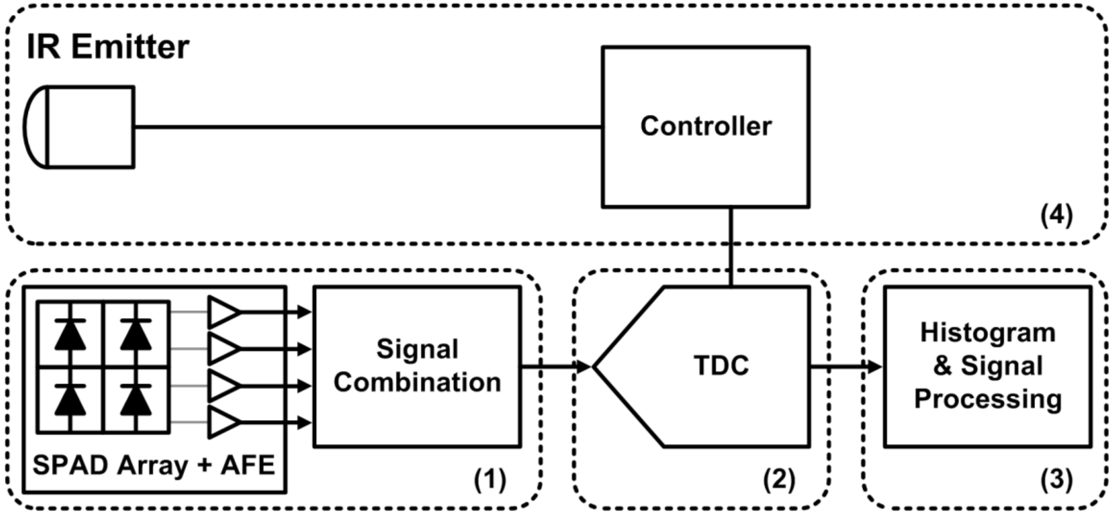
- Detect direct time difference (TOF)



- Depth = c/2 × TOF (c ≈ 3×10^8 [m/s])
- The sensor's receiver utilizes a single-photon avalanche diode (SPAD)

A Direct TOF Sensor

- 4 Primary Components

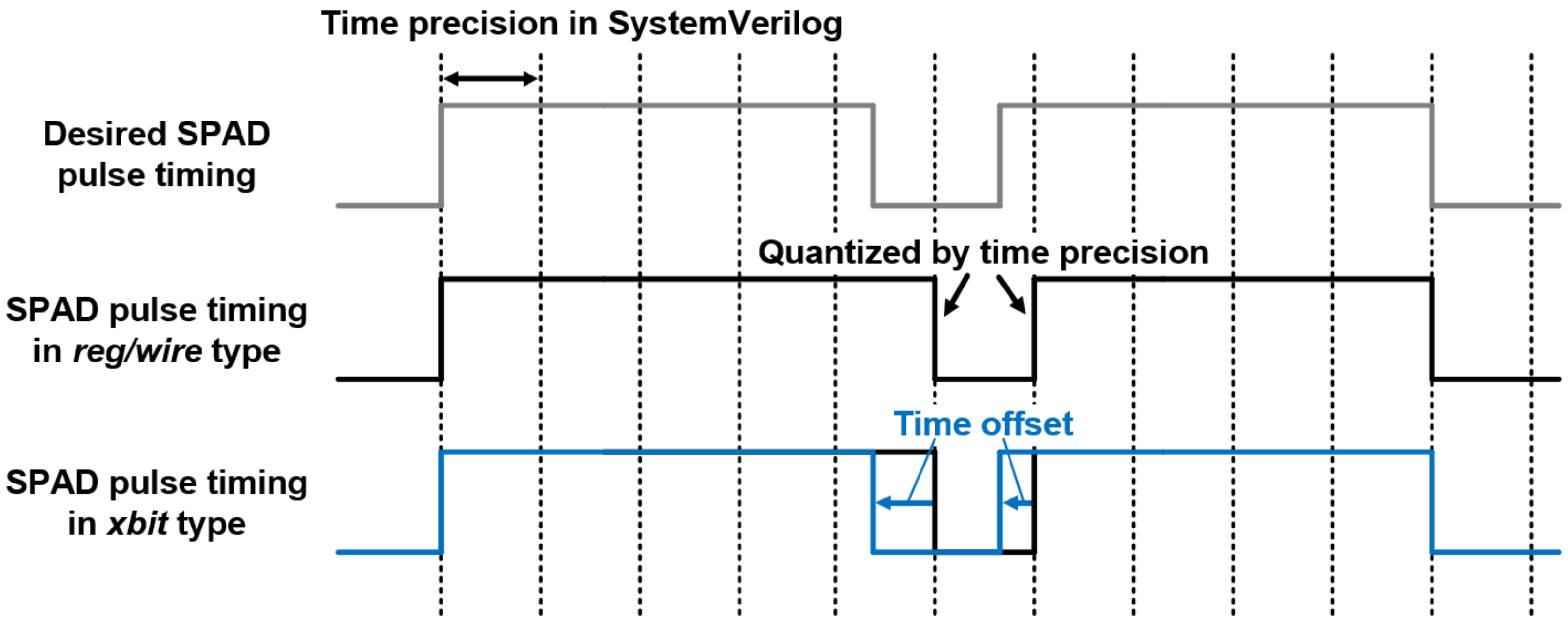


- (1) SPAD array, analog front-end (AFE) and signal combiner
- (2) Time-to-digital converter (TDC)
- (3) Histogramming and digital signal processing
- (4) Timing controller

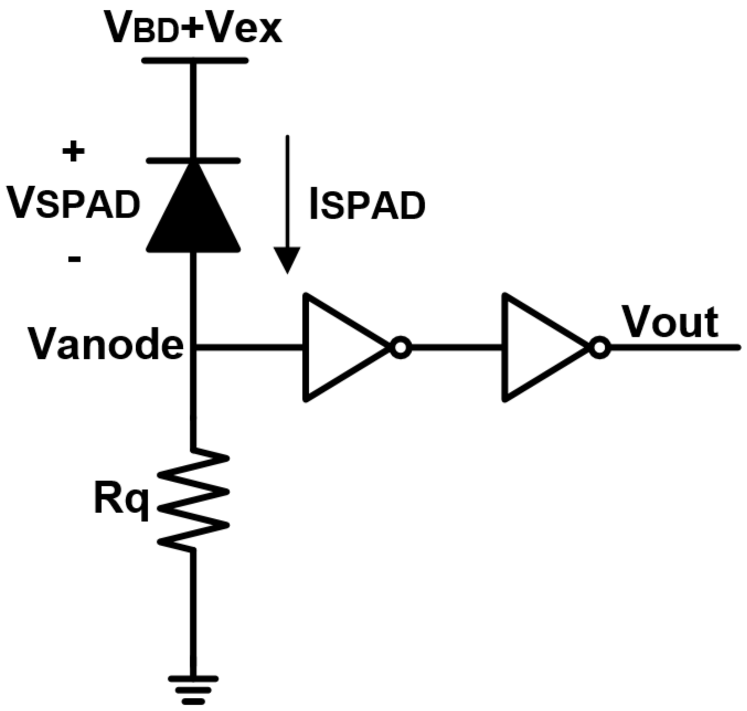
- To predict the accuracy of the TOF sensor
  - we need a SPAD model that simulates the physical and statistical characteristics
  - A simulation platform is essential

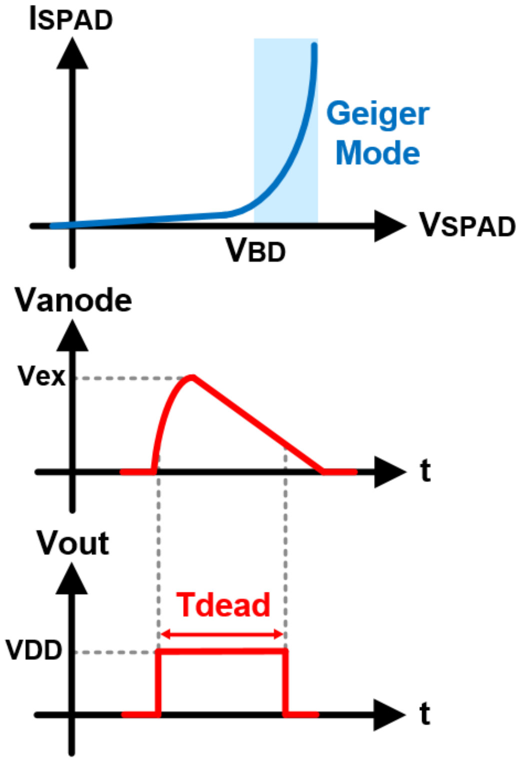
XMODEL: Accurate Simulation in SystemVerilog

- XMODEL: plug-in extension to SystemVerilog developed by Scientific Analog
- ***xbit*** type can express precise timing information without being limited by the timestep of SystemVerilog simulation



SPAD & AFE Operation



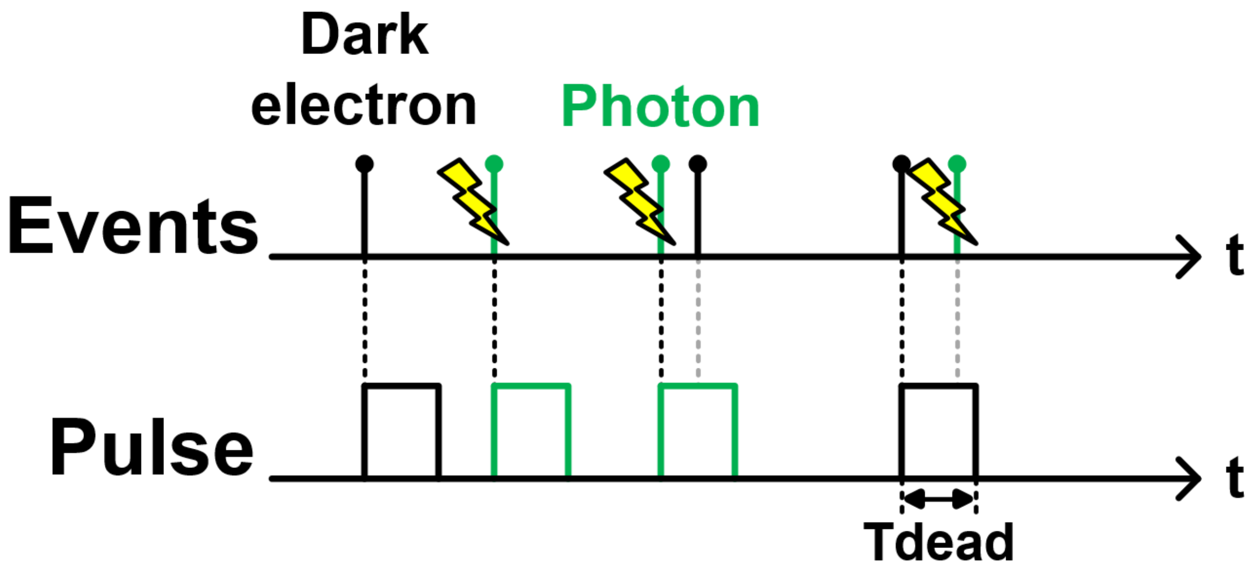


- SPAD functions in Geiger mode

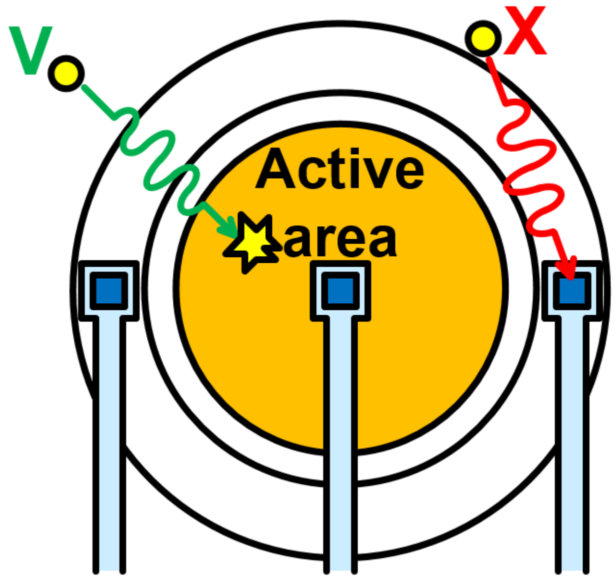
- Upon the incidence of a photon on the SPAD, a digital pulse is generated
- The width of the digital pulse is referred to as the SPAD dead-time (Tdead)

SPAD Characteristics

- Dark count rate (DCR), Dead-time (Tdead)
- Photon detection probability (PDP)



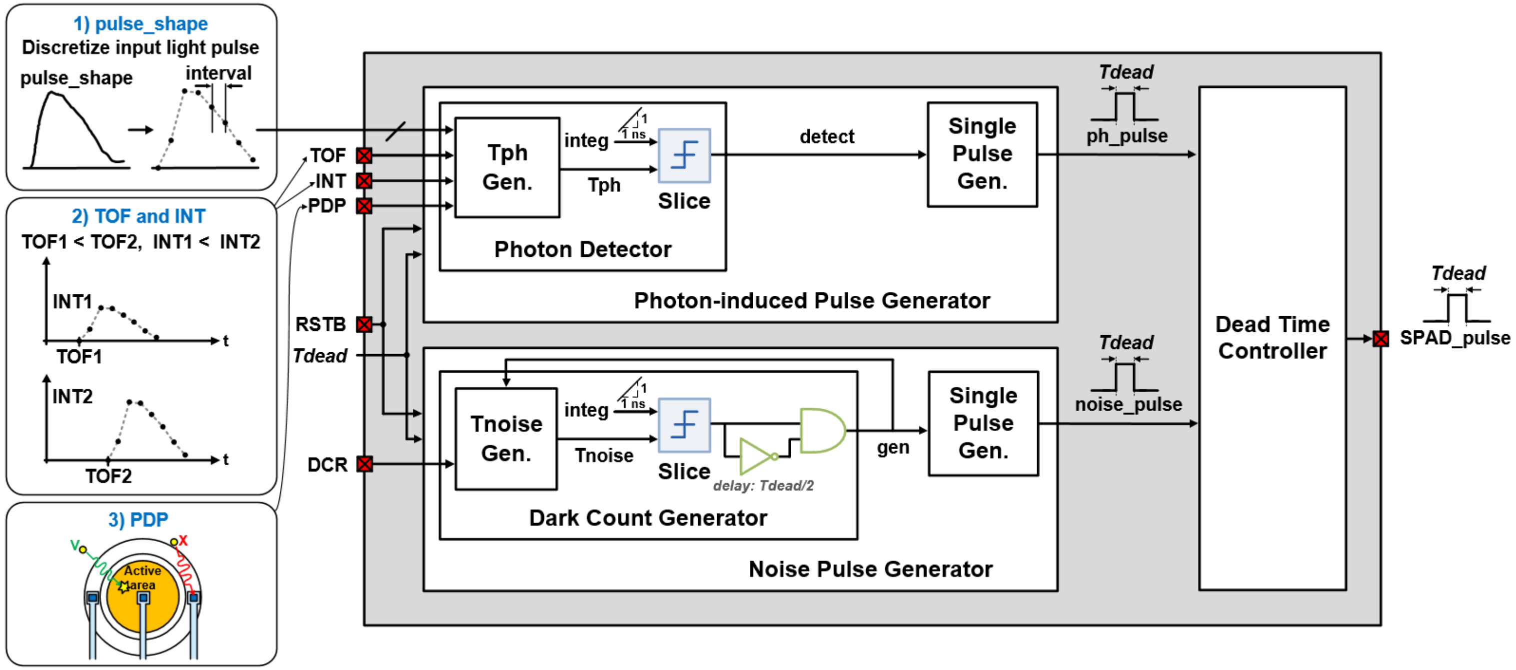
DCR, Tdead



PDP

SPAD Modeling

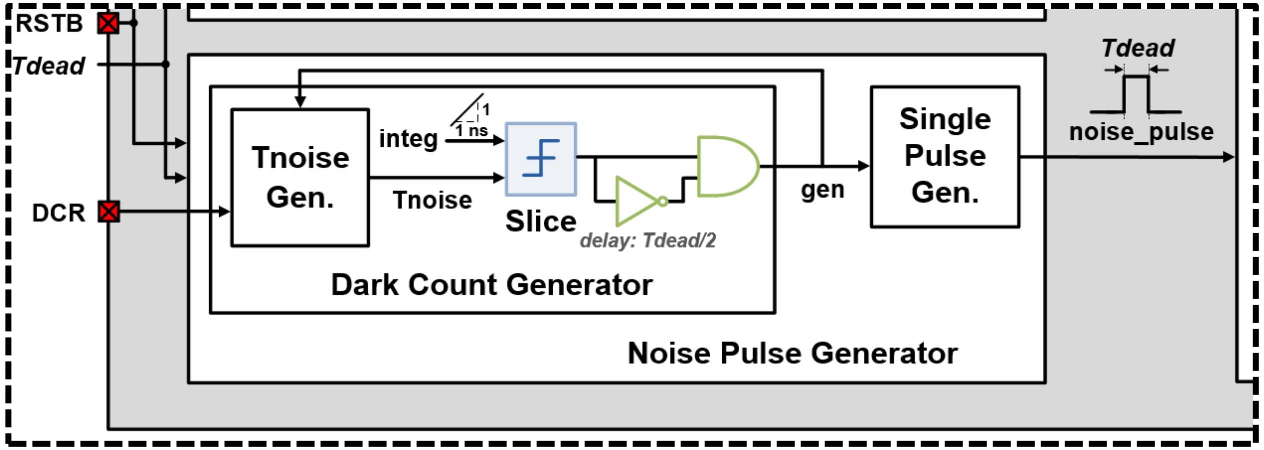
- Calculates the generation time of noise pulse and photon-induced pulse through a Poisson process
- Generates output pulses with a user-defined pulse width of *Tdead*

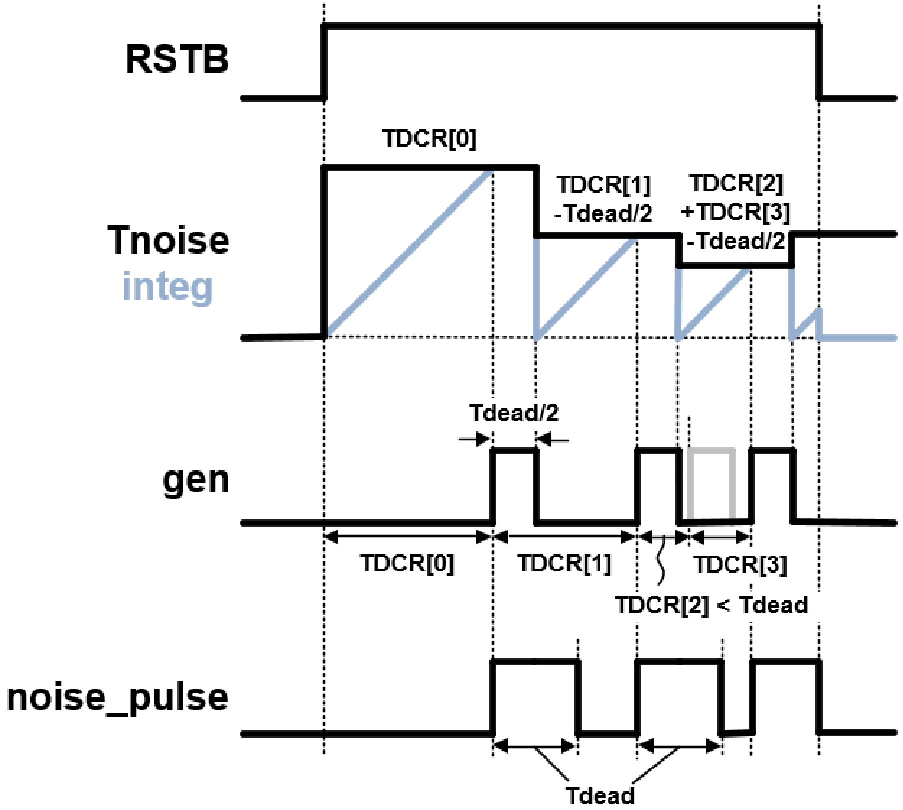


- Proposed SPAD modeling performs:
  - Noise pulse generation
  - Photon-induced pulse generation
  - Dead time control

Noise Pulse Generator

- Tnoise Generator generates the randomized timing of noise pulse initiation, *Tnoise*
  - The mean frequency of noise pulse generation is *DCR*
- Single Pulse Generator creates *noise_pulse* with a width of *Tdead*





Tnoise Generator

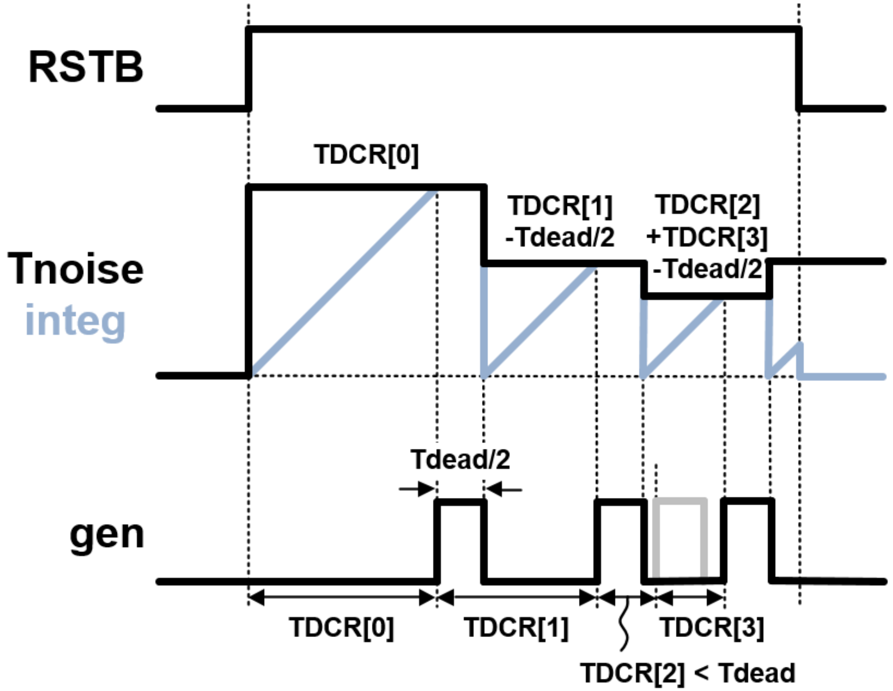
- *Tnoise* is ascertained from the Poisson process with a rate of λ
- The probability that the first dark count generation time, X_1, exceeds time t
  - $P(X_1 > t) = P(no\ dark\ count\ generation\ between\ 0\ and\ t) = e^{-\lambda t}$
- The probability that the first dark count generation occurs within the time t
  - $P(X_1 \leq t) = 1 - e^{-\lambda t}$
- The interarrival time $t = \frac{\ln(1 - P(X_1 \leq t))}{-\lambda}$
  - t → *Tnoise*
  - $P(X_1 \leq t)$ → *rand_uniform(0,1)* function
  - λ → *DCR*

Tnoise Generator (2)

- *Tnoise* = $\frac{\ln(1-rand\_uniform(0,1)\ )}{-DCR}$
- *Tnoise* is updated every time *gen* has a falling edge
  - Considering *Tnoise* ≥ *Tdead*

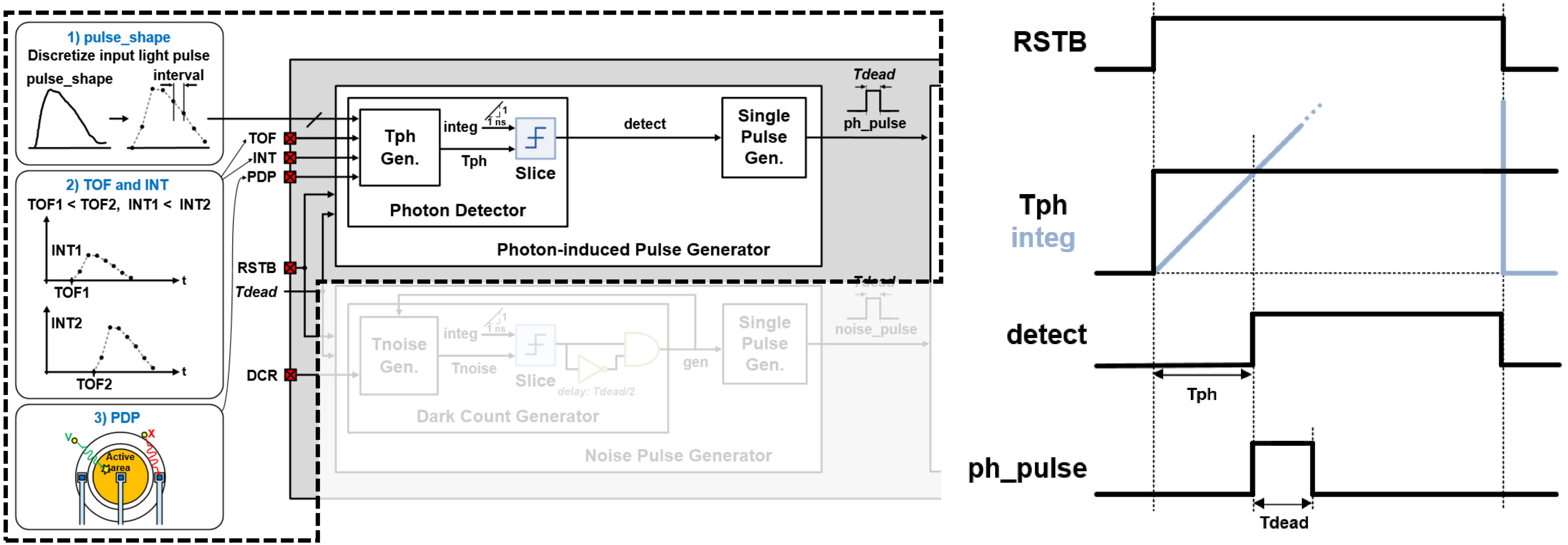
```
module Tnoise_gen
…
//For initial Tnoise (TDCR)
always @(posedge RSTB) begin
   random_dark = rand_uniform(0,1);
   …
  TDCR = $ln(1-random_dark) / (-DCR);
end
//For updating Tnoise when gen has a falling edge
always @(negedge gen) begin
   TDCR = 0;  random_dark = rand_uniform(0,1);
```

```
…
   TDCR = $ln(1-random_dark) / (-DCR);
   //For considering dead-time (Tdead)
   while(TDCR < Tdead) begin
      random_dark = rand_uniform(0,1);
      …
      TDCR = TDCR + $ln(1-random_dark) / (-DCR);
   end
   TDCR = TDCR – (Tdead/2);
end
assign Tnoise = TDCR;
endmodule
```



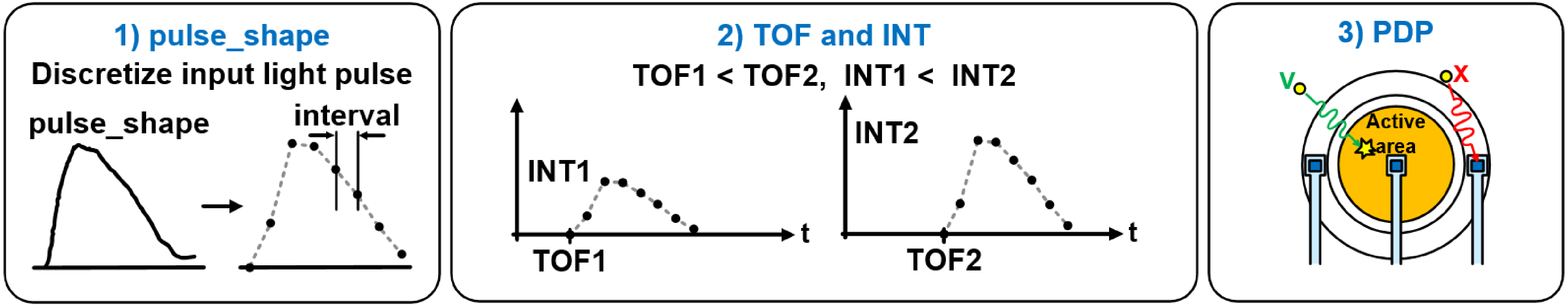
Photon-induced Pulse Generator

- Tph Generator calculates the photon-induced pulse generation time, *Tph*
  - Inputs: the light pulse shape (*pulse_shape*), *TOF*, light intensity (*INT*), and *PDP*



Tph Generator

- It takes the light pulse shape (*pulse_shape*), photon arrival time (*TOF*), light intensity (*INT*), and photon detection probability (*PDP*) as inputs



Tph Generator (2)

- 1. *pulse_shape_disc* is obtained by discretizing the input *pulse_shape*
- 2. *pulse_shape_int* is achieved by normalizing *pulse_shape_disc* and multiplying it by *INT*
  - *Tres*: resolution time of *pulse_shape_int*

```
module Tph_gen
…
always @(posedge RSTB) begin
  signal_sequence = 0;    //Photon detection time within input light pulse
  Tph = 0;                //Photon detection time (= TOF + signal_sequence)
  pulse_sum = 0;          //Sum of discretized pulse_shape data (pulse_shape_disc)

  //For discretizing pulse_shape
  //(Resolution time of pulse_shape_disc = (interval * 0.02) ns)
  for(i=0; i<(pulse_len/interval); i++) begin
    pulse_shape_disc[i+1] = pulse_shape[i*interval+int'(interval/2)+1];
  end
```

```
  //For normalizing pulse_shape_disc
  for(j=0; j<(pulse_len/interval)+1; j++) begin
    if(pulse_shape_disc[j]<0) pulse_shape_disc[j] = 0;
    pulse_sum = pulse_sum + pulse_shape_disc[j];
  end
  for(k=0; k<(pulse_len/interval)+1; k++) begin
    pulse_shape_int[k] = pulse_shape_disc[k] / pulse_sum * INT;
  end
  Tres = 0.02 * interval;  //Tres: Resolution time of pulse_shape_int
```

Tph Generator (3)

- 3. Use the cumulative distribution function (cdf) of the Poisson process

  to calculate at which point within *pulse_shape_int* (λ) the SPAD reacts.

  - cdf = $\sum_{k=0}^{n} \frac{e^{-\lambda}\lambda^{k}}{k!}$
  - The value of *k* increments by one until *cdf* > *thres_cdf* (=*rand_uniform(0,1))*
  - The derived *k*: the number of photons successfully reaching the SPAD

```
//For modeling photon arrival and SPAD avalanche
  for(i=0; i<(pulse_len/interval)+1; i++) begin
    k = 0;  cdf = 0;  kfactorial = 1;   thres_cdf = rand_uniform(0,1);
    while (1) begin
      cdf = cdf + $exp(-pulse_shape_int[i]) * $pow(pulse_shape_int[i], k) / kfactorial;
       if(cdf <= thres_cdf) begin
```

Tph Generator (4)

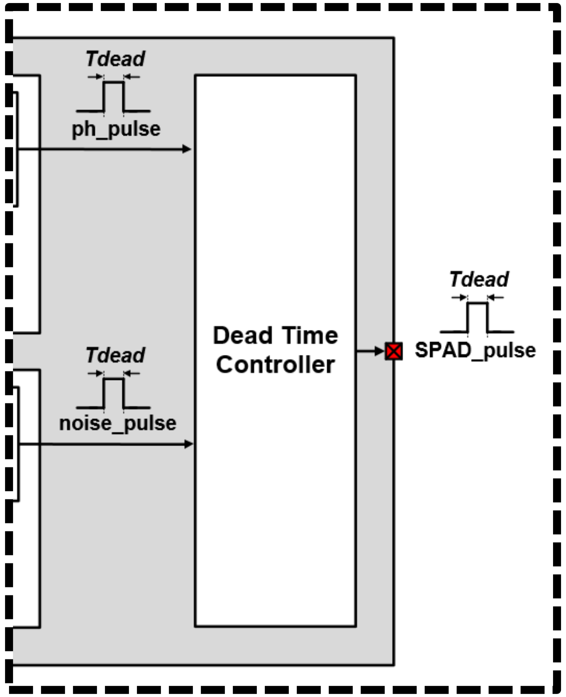
- 4. Given the SPAD's PDP, a determination is made as to whether each photon's arrival instigates an avalanche
  - If the derived *k* = 2, the comparison between *thres_pdp* and *PDP* is performed twice
  - If the PDP criterion is met for the first time at the *i*-th datum of *pulse_shape_int*, *Tph* = *TOF* + *i* × *Tres* (*Tres*: resolution time of *pulse_shape_int*)

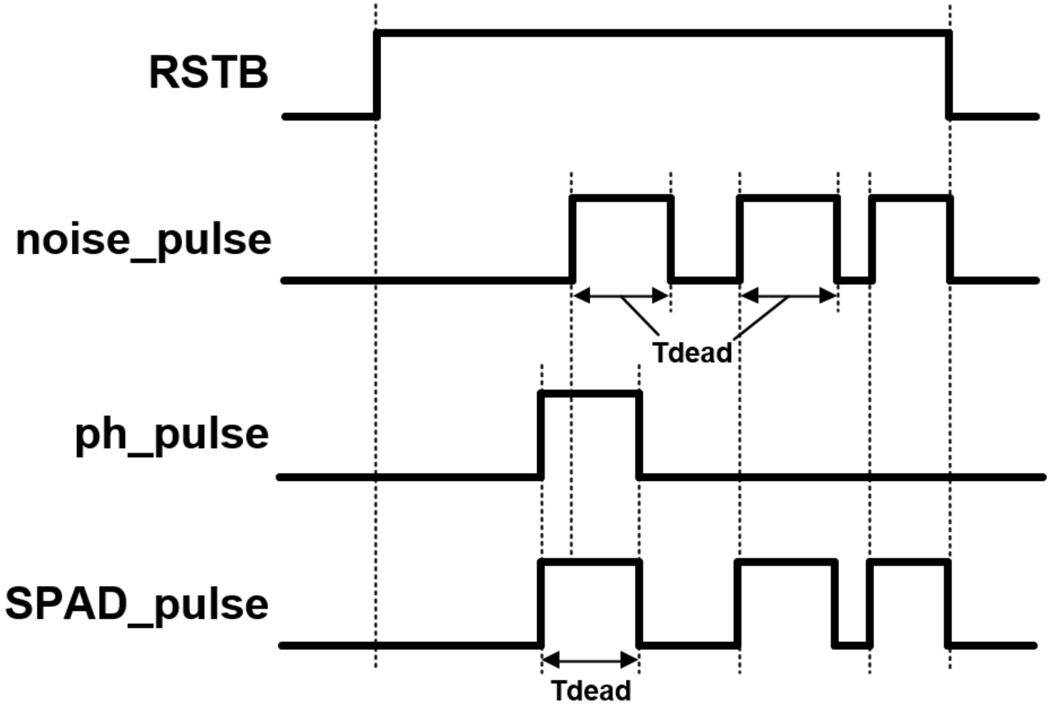
```
//For considering PDP of the SPAD
            thres_pdp = rand_uniform(0,1);
            if(PDP > thres_pdp) begin
               signal_sequence = i * Tres;
               break;
            end
            k = k+1;  kfactorial = kfactorial * k;
         end
         else break;
      end
```

```
      if(signal_sequence != 0) break;
   end
   if(signal_sequence != 0) Tph = TOF + signal_sequence;
end
endmodule
```

Dead Time Controller

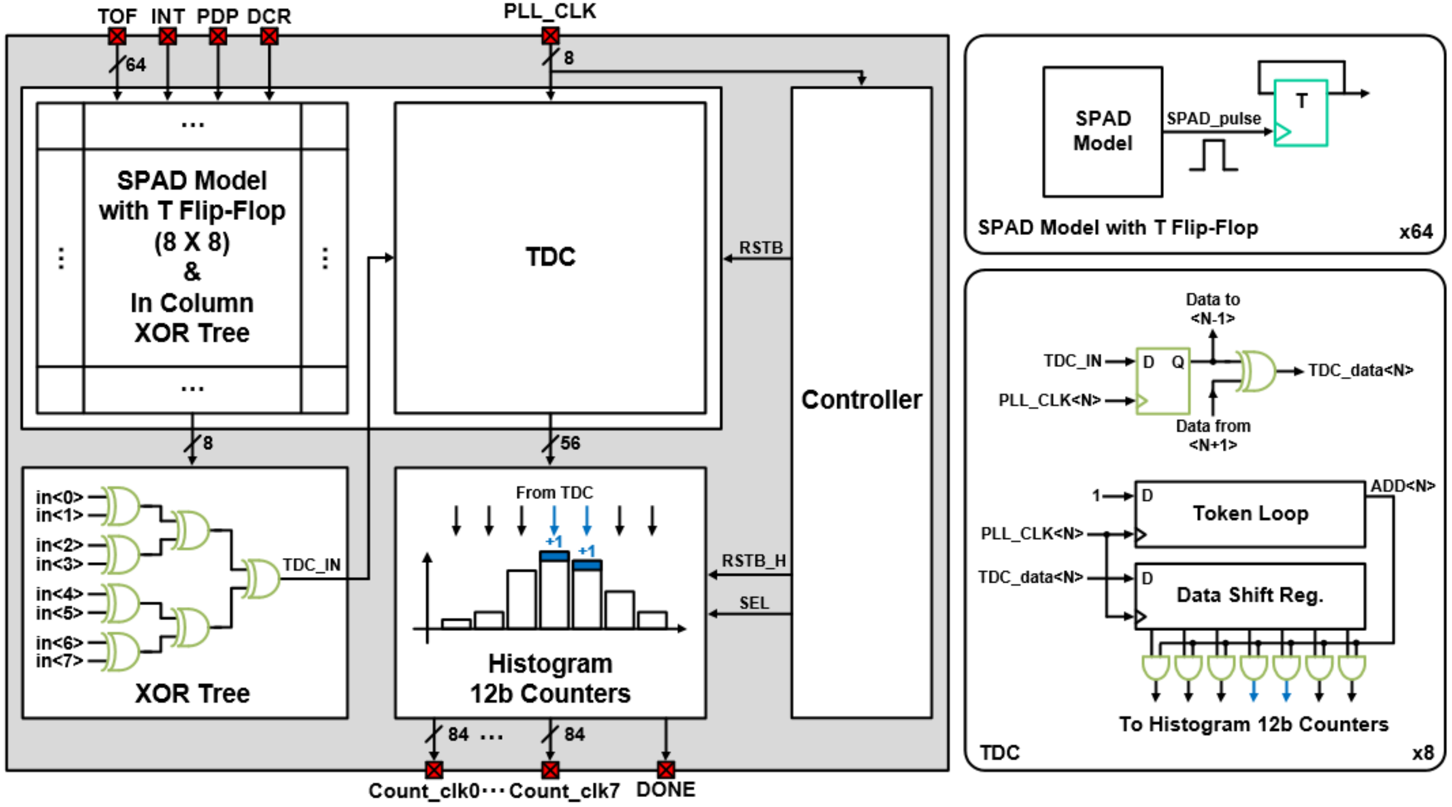
- Compares *noise_pulse* and *ph_pulse*
- Outputs the final *SPAD_pulse*





SPAD-Based Sensor Modeling

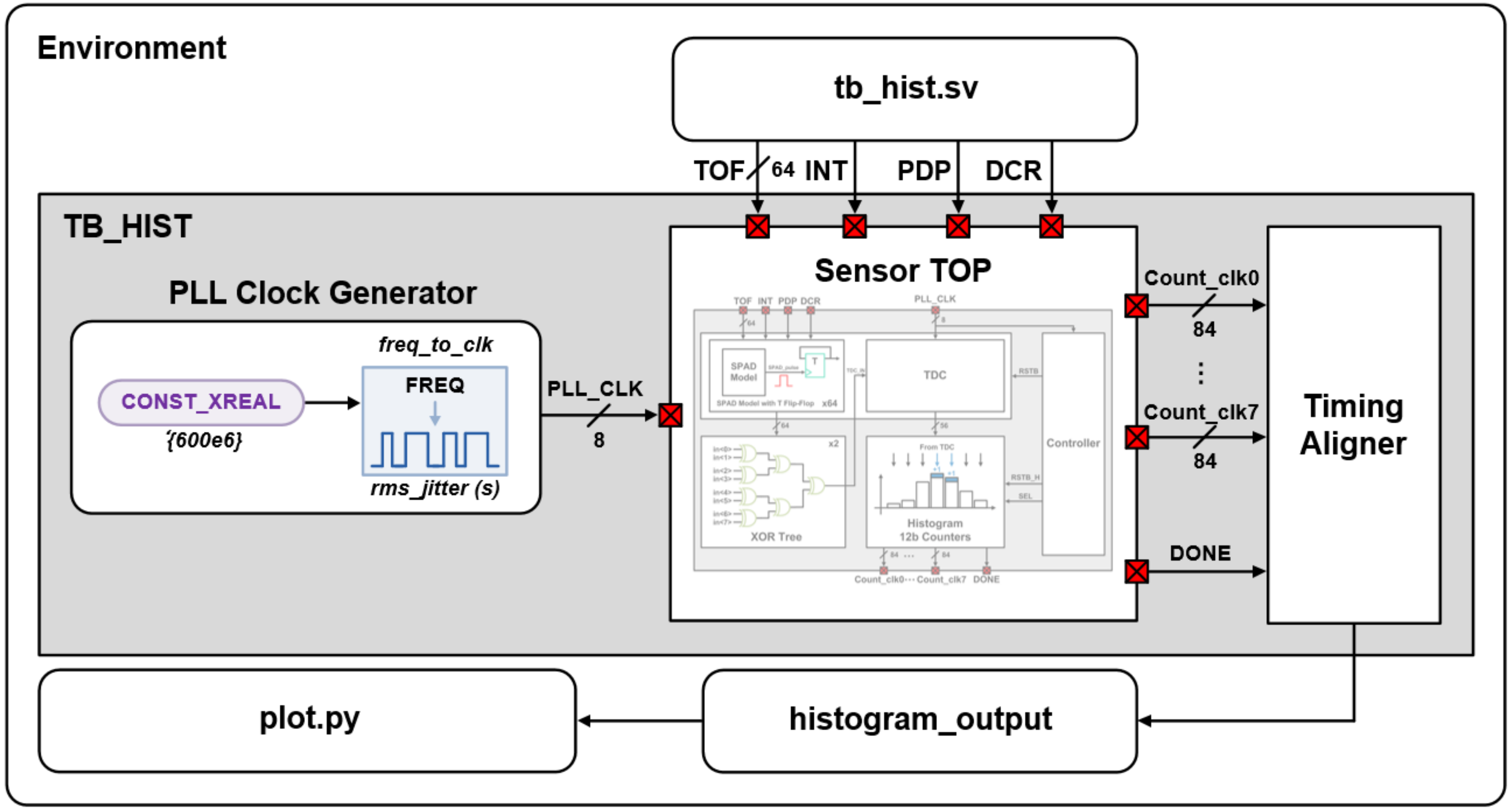
- The overall block diagram of a SPAD-based sensor configured as a proximity sensor



- *TDC_IN* toggles whenever 64 SPADs react
- TDC digitizes the timing of *TDC_IN* toggles using *PLL_CLK*
- Using ripple counter method, up to 4095 measurements can be made
- Controller adjusts the overall system operation timing

[T. Al Abbas, et al., 2018]

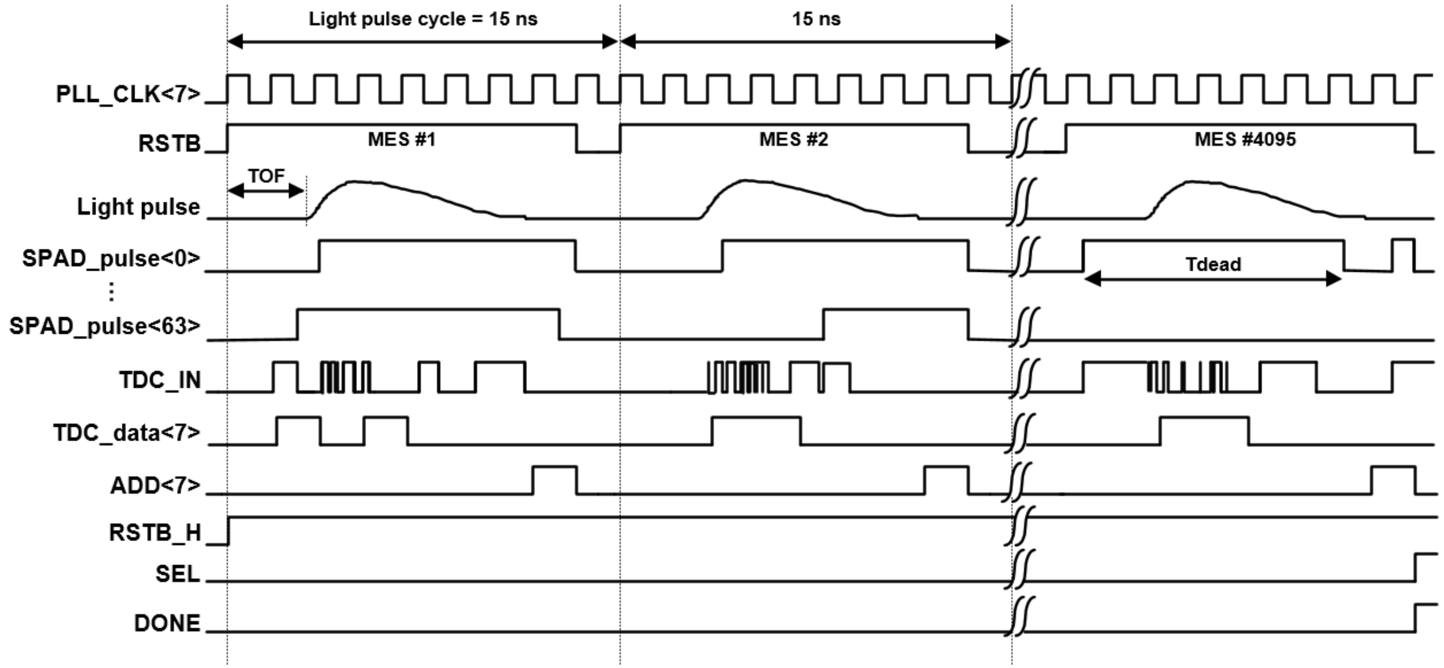
Testbench for a SPAD-Based Sensor Simulation



- *PLL_CLK* frequency: 600MHz
  - TDC time resolution: 208 ps
  - RMS jitter was set to 6 ps
- Timing Aligner
  - rearranges the histogram outputs (*Count_clk0~7*) according to their actual timing
  - saves the outputs in *histogram_output*

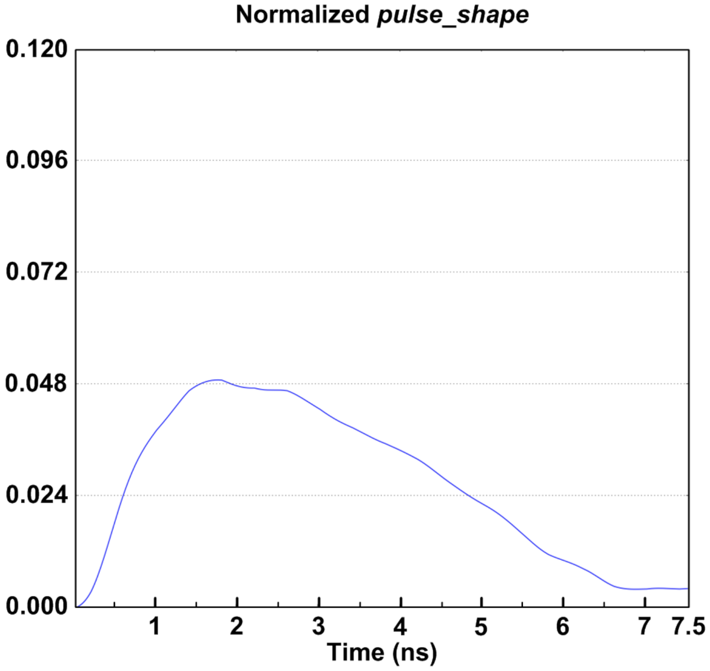
Timing Diagram of the SPAD-Based Sensor Modeling

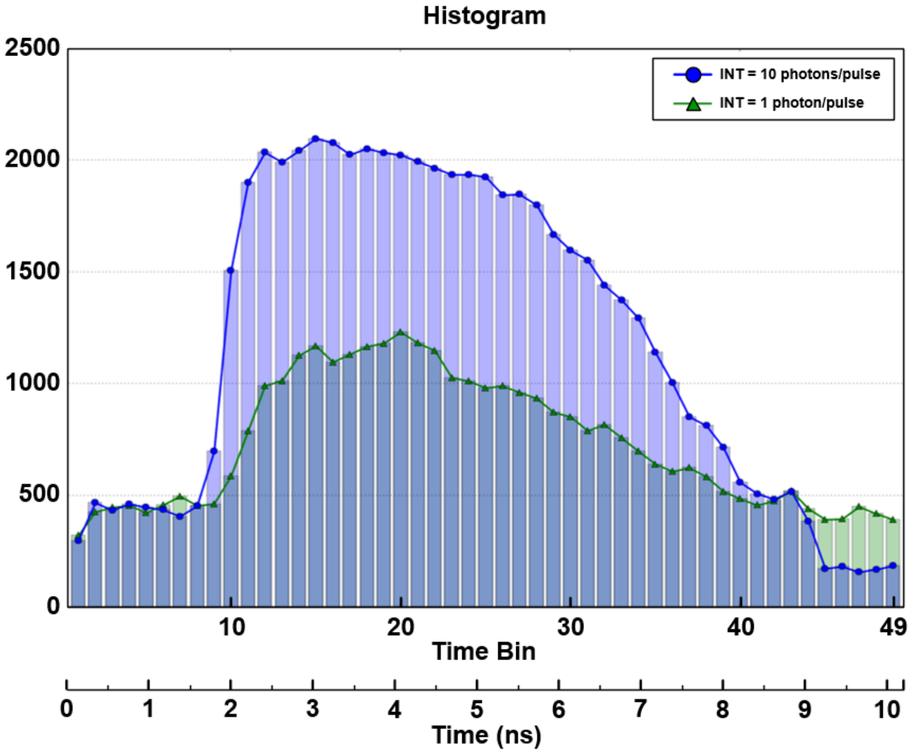
- Light pulse cycle was set to 15 ns, including a reset timing of 1.66 ns
  - The sensor can measure up to a distance of about 153 cm
  - A total convergence time of 65 us is required to measure TOF 4095 times



Histogram Results

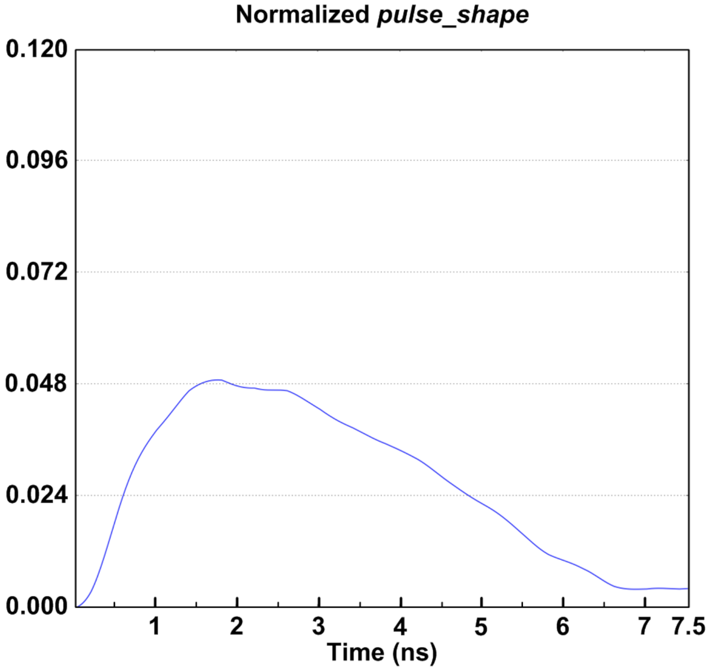
- Light pulse with a dispersed shape (7.5 ns width)
  - Common inputs: TOF = 1.5 ns, PDP = 0.1, DCR = 0.01 counts/ns, Tdead = 10 ns

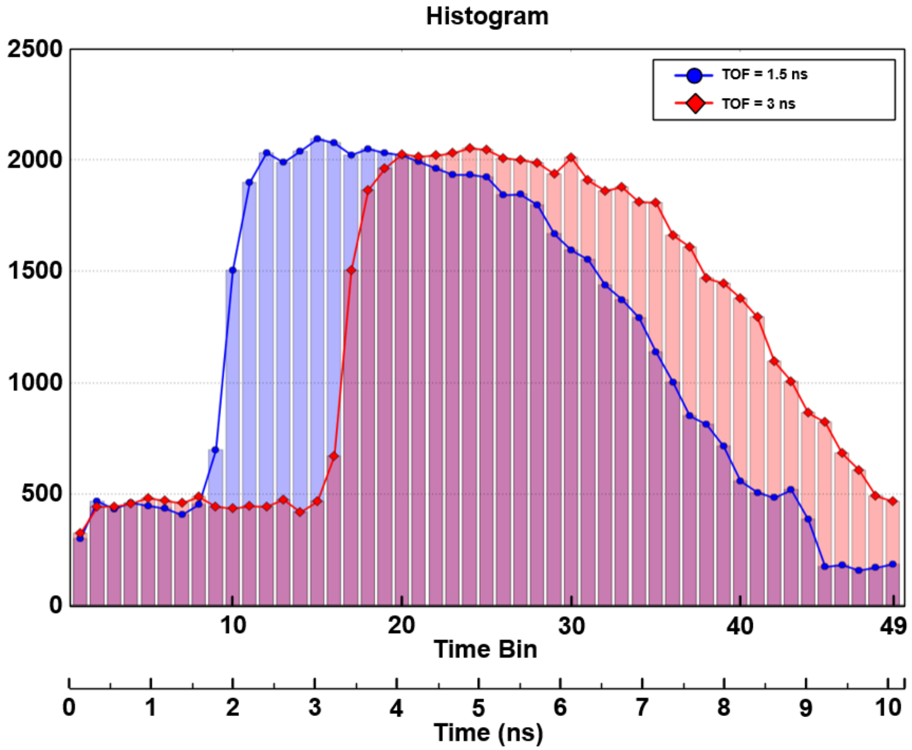




Histogram Results

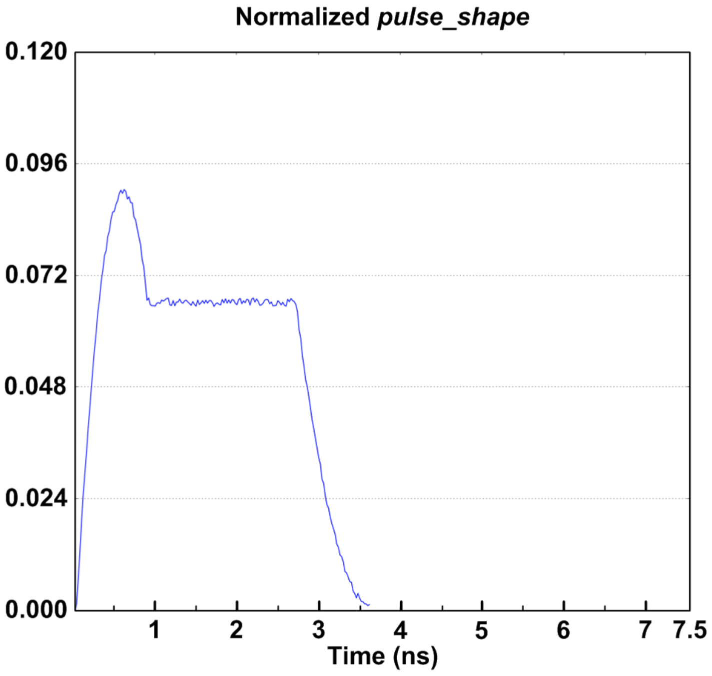
- Light pulse with a dispersed shape (7.5 ns width)
  - Common inputs: INT = 10 photons/pulse, PDP = 0.1, DCR = 0.01 counts/ns, Tdead = 10 ns

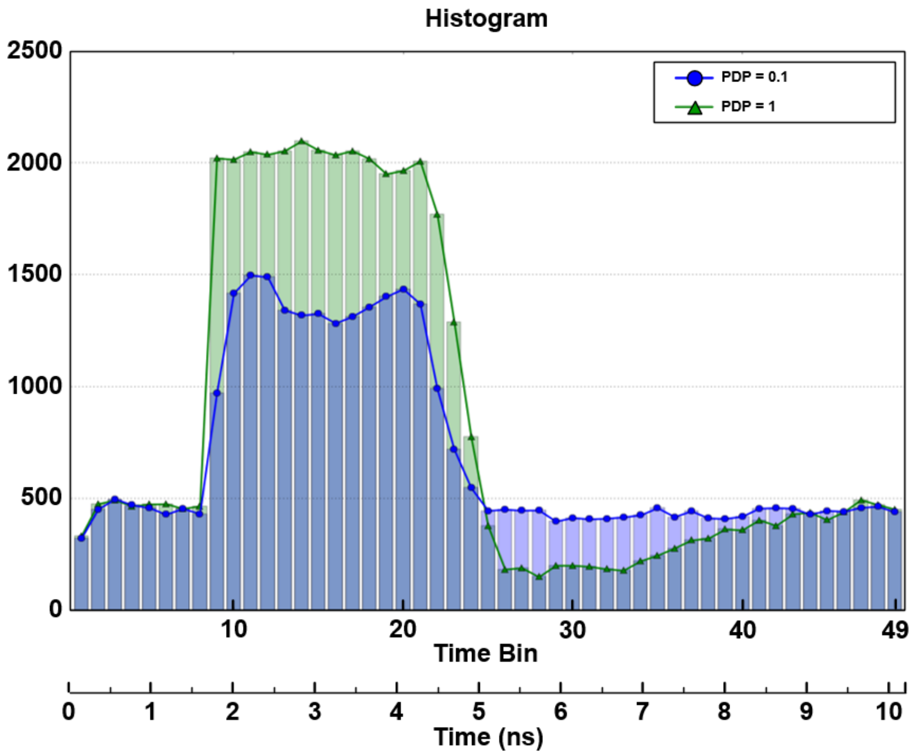




Histogram Results

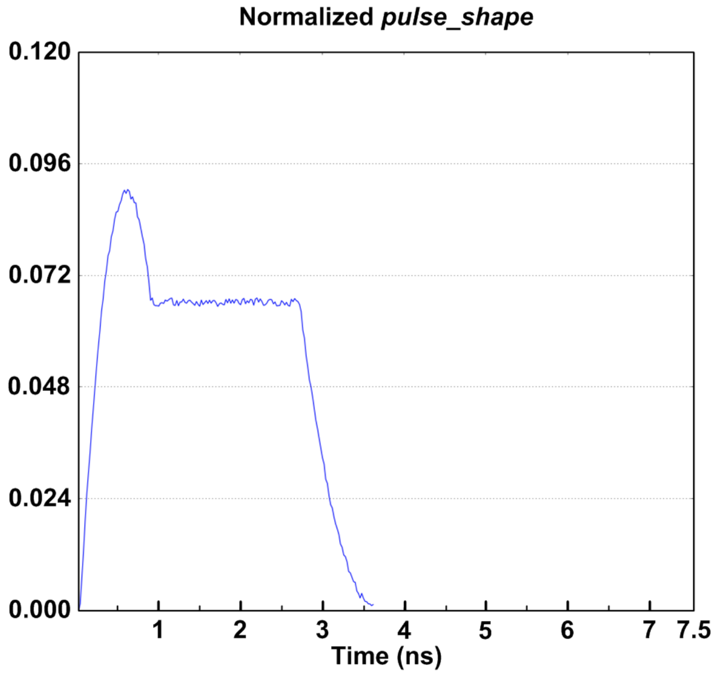
- Light pulse with relatively sharp shape (3.6 ns width)
  - Common inputs: INT = 1 photon/pulse, TOF = 1.5 ns, DCR = 0.01 counts/ns, Tdead = 5 ns

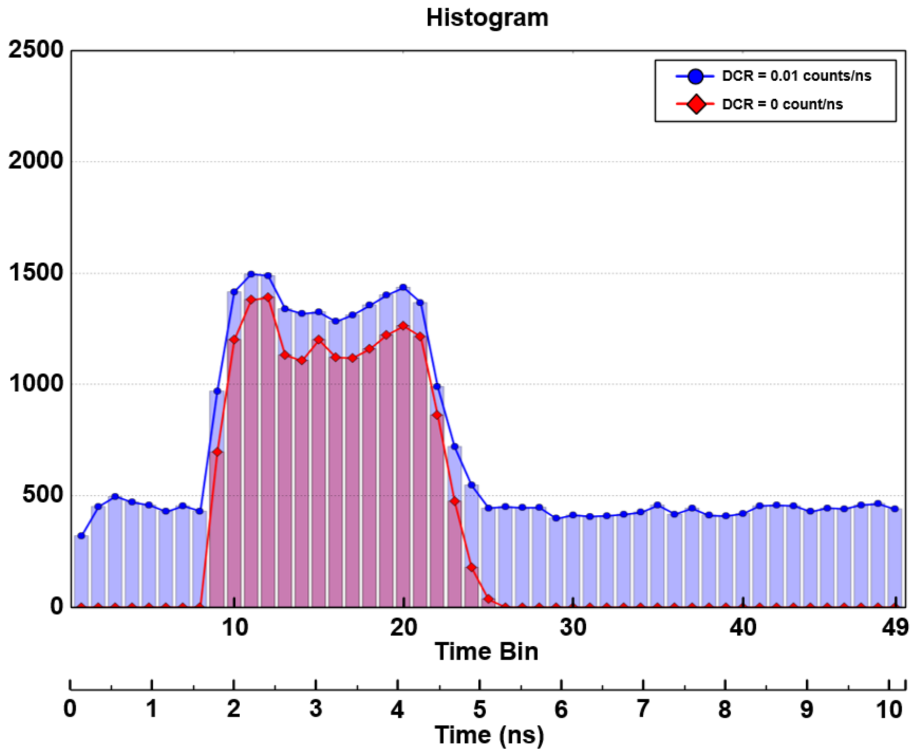




Histogram Results

- Light pulse with relatively sharp shape (3.6 ns width)
  - Common inputs: INT = 1 photon/pulse, TOF = 1.5 ns, PDP = 0.1, Tdead = 5 ns





Summary

- This work demonstrated the feasibility of system-level simulation of SPAD-based sensors in SystemVerilog using XMODEL primitives

- By utilizing the proposed statistical behavioral model of the SPAD, the entire sensor system can be verified

- A 1.5-meter range proximity sensor completes its simulation in just 4.6 minutes, yielding 4095 histogram data points

Questions

- Thank you