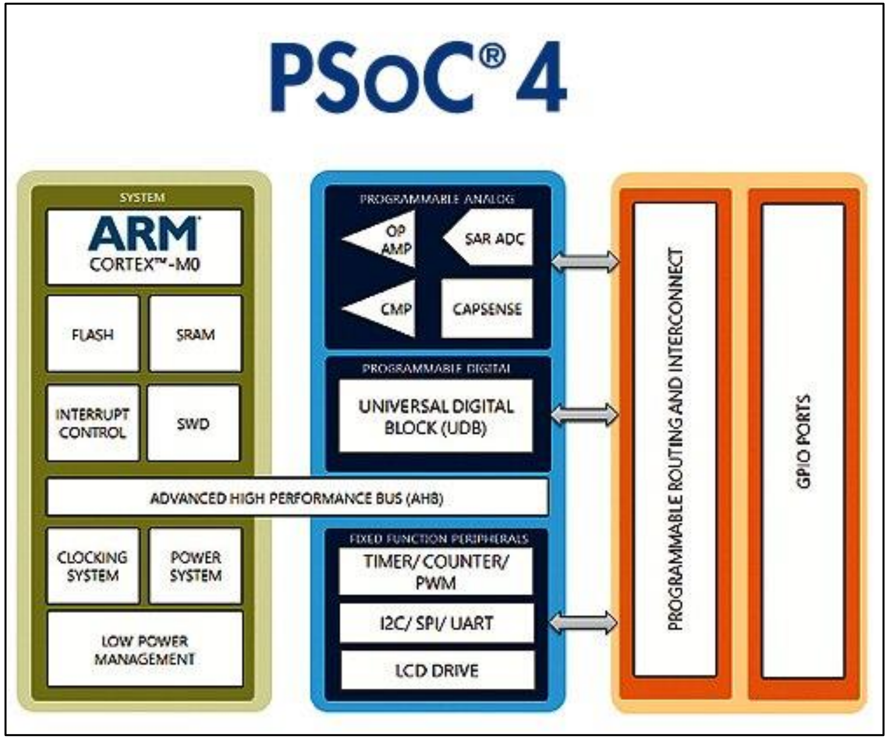
Chapter-3

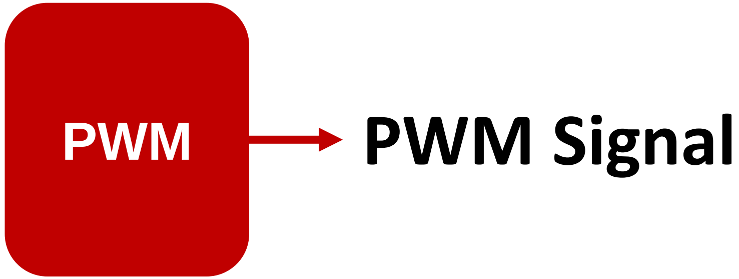
Digital Blocks

PWM

PWM

Pulse Width Modulator

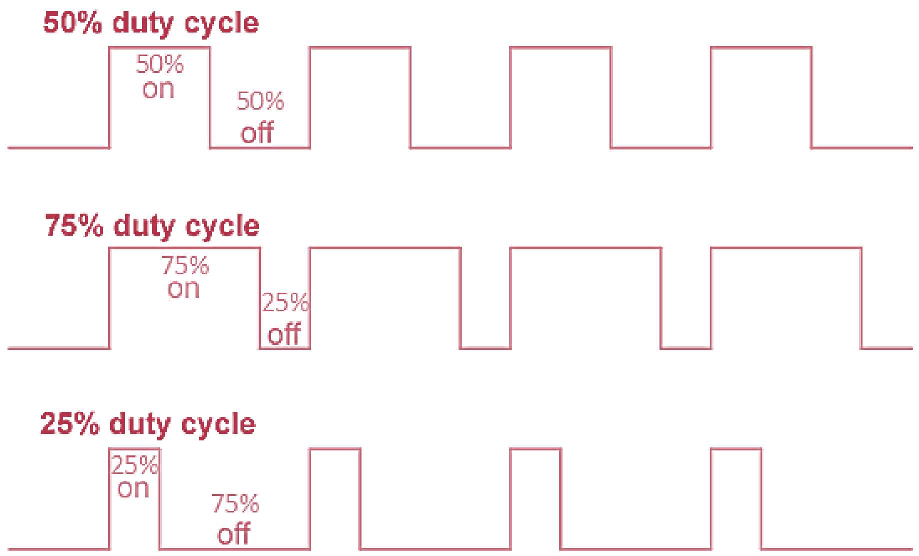




https://community.element14.com/products/devtools/technicallibrary/w/documents/11058/cypress-psoc-4-system-on-chip-overview

PWM Signal

“A Square Wave Signal with a Specified Duty Cycle”



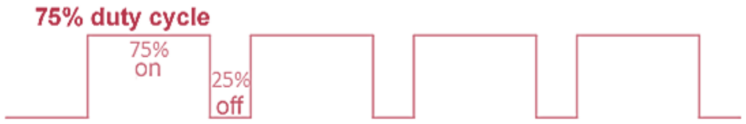
$$\text{Duty Cycle} = \frac{T_{ON}}{T_{ON} + TOFF}$$

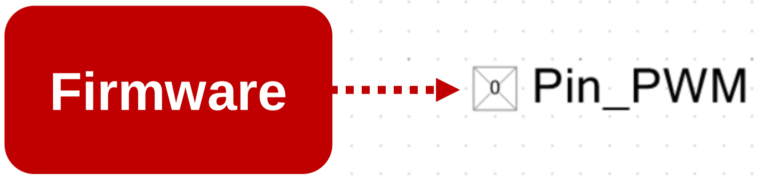
$$\text{Freq} = \frac{1}{T_{PERIOD}} = \frac{1}{T_{ON} + TOFF}$$

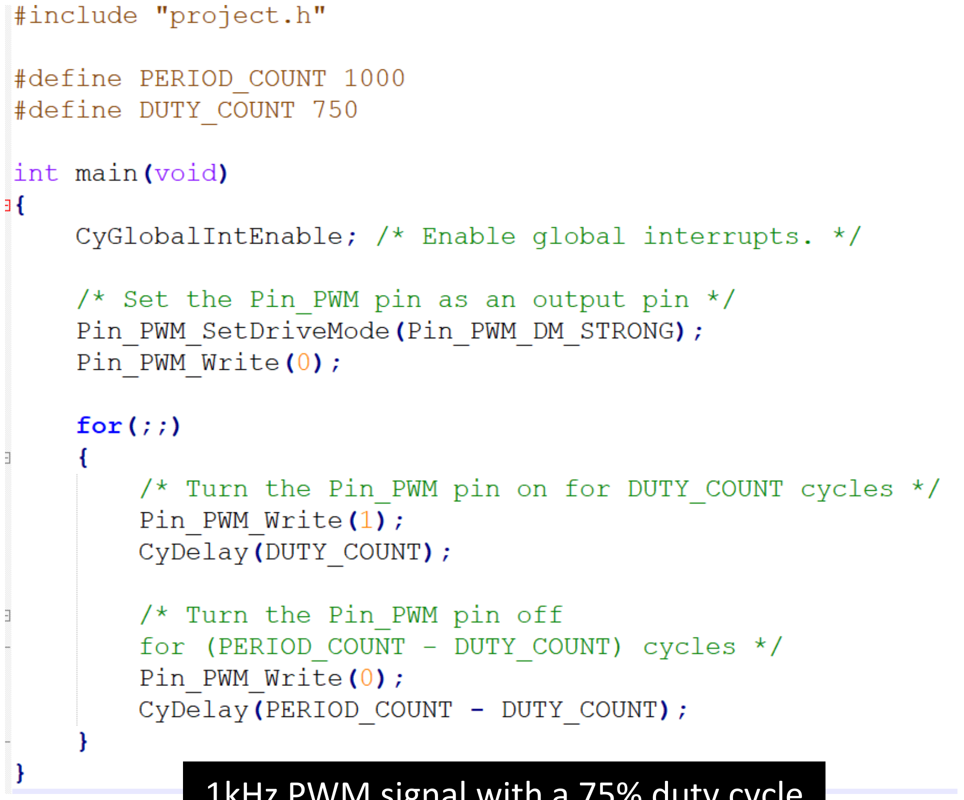
$$\text{Amplitude} = V_{HIGH} - V_{LOW}$$

https://en.wikipedia.org/wiki/Pulse-width_modulation

Generation (Method #1)





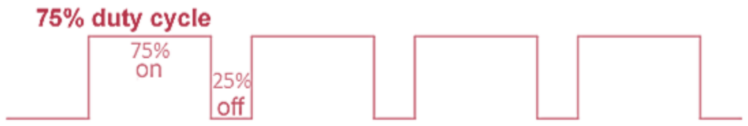


```
#include "project.h"

#define PERIOD_COUNT 1000
#define DUTY_COUNT 750

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Set the Pin_PWM pin as an output pin */
    Pin_PWM_SetDriveMode(Pin_PWM_DM_STRONG);
    Pin_PWM_Write(0);

    for(;;)
    {
        /* Turn the Pin_PWM pin on for DUTY_COUNT cycles */
        Pin_PWM_Write(1);
        CyDelay(DUTY_COUNT);

        /* Turn the Pin_PWM pin off
        for (PERIOD_COUNT - DUTY_COUNT) cycles */
        Pin_PWM_Write(0);
        CyDelay(PERIOD_COUNT - DUTY_COUNT);
    }
}
```

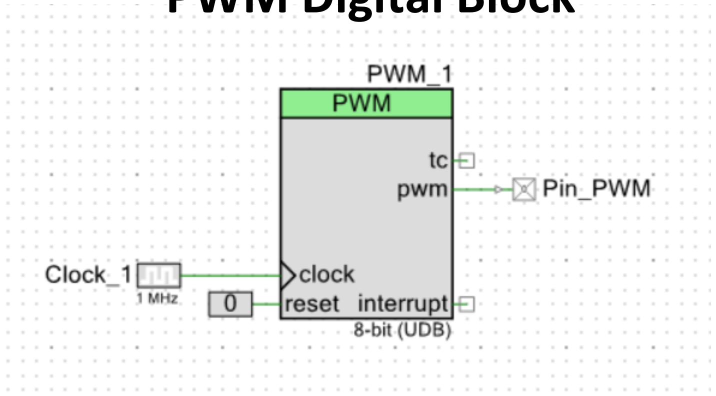
1kHz PWM signal with a 75% duty cycle

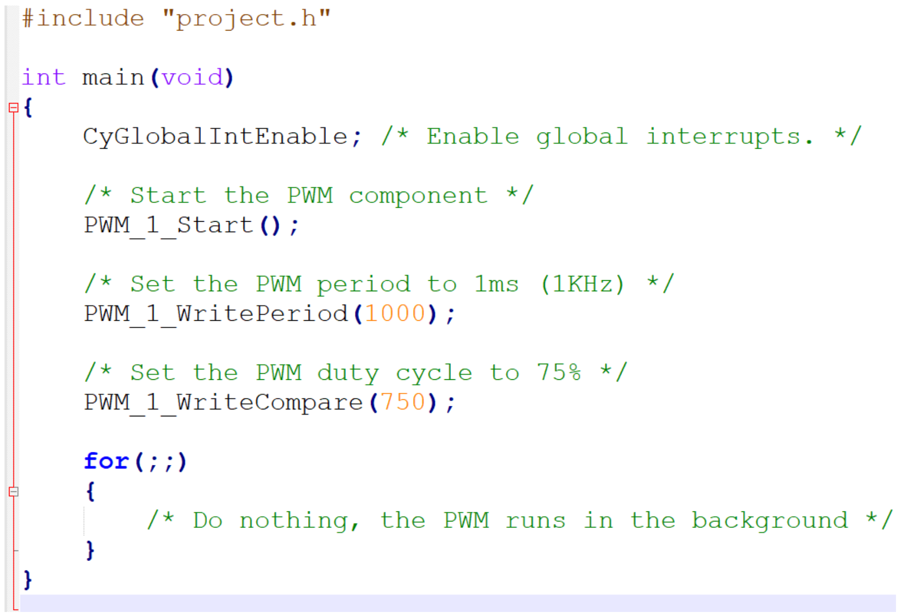
https://en.wikipedia.org/wiki/Pulse-width_modulation

Generation (Method #2)



PWM Digital Block

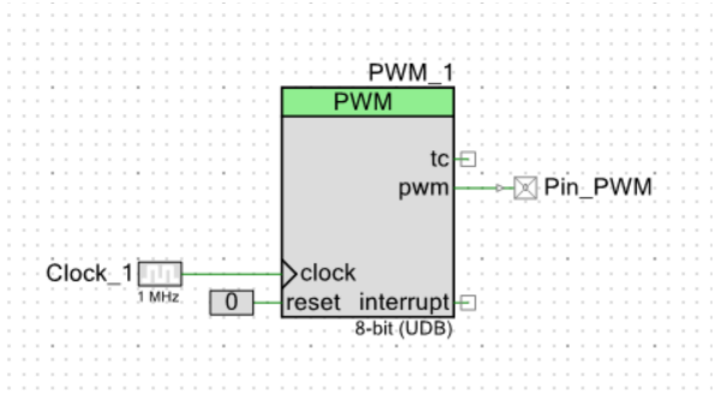


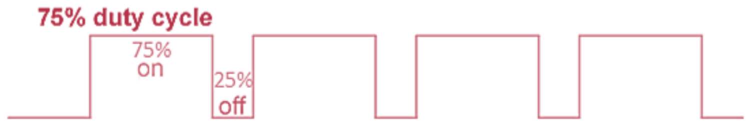


```
#include "project.h"

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Start the PWM component */
    PWM_1_Start();

    /* Set the PWM period to 1ms (1KHz) */
    PWM_1_WritePeriod(1000);

    /* Set the PWM duty cycle to 75% */
    PWM_1_WriteCompare(750);

    for(;;)
    {
        /* Do nothing, the PWM runs in the background */
    }
}
```

https://en.wikipedia.org/wiki/Pulse-width_modulation

Component Settings





$$\mathbf{Freq} = \frac{1}{T_{\mathrm{PERIOD}}} = \frac{1}{\mathbf{Period} * \mathbf{Tclock\_1}}$$

$$\mathbf{Duty\ Cycle} = \frac{\mathbf{Compare}}{\mathbf{Period}}$$

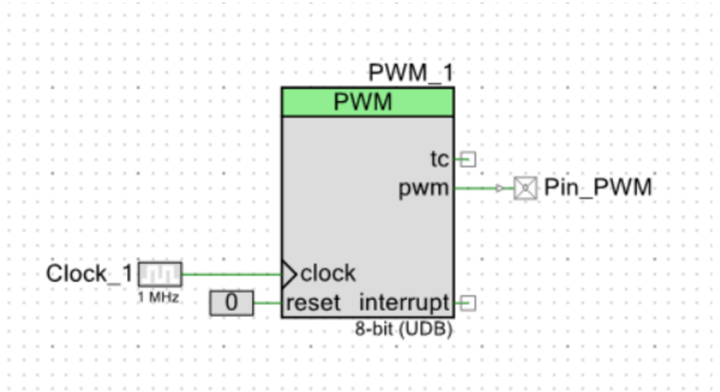
PWM Bits = 8bit (or 16bit)

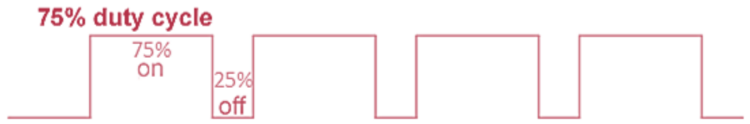
PWM Period = 0-255 (or 0 to 65535)

PWM Compare = 0-255 (or 0 to 65535)

https://en.wikipedia.org/wiki/Pulse-width_modulation

E.g., Settings





$$\text{Freq} = \frac{1}{T_{\text{PERIOD}}} = \frac{1}{\text{Period} * \text{Tclock}\_1} \quad ?$$

$$\text{Duty Cycle} = \frac{\text{Compare}}{\text{Period}} \quad ?$$

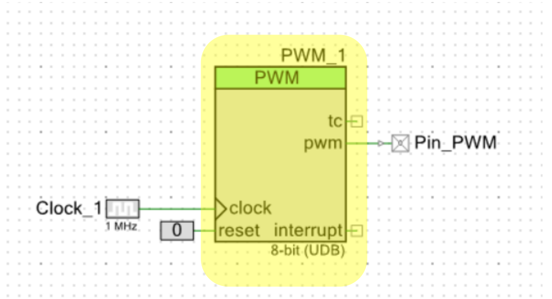
PWM Bits = 8bit

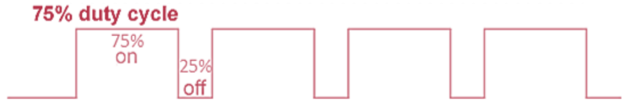
PWM Period = 200

PWM Compare = 50

https://en.wikipedia.org/wiki/Pulse-width_modulation

Verilog 8bit PWM



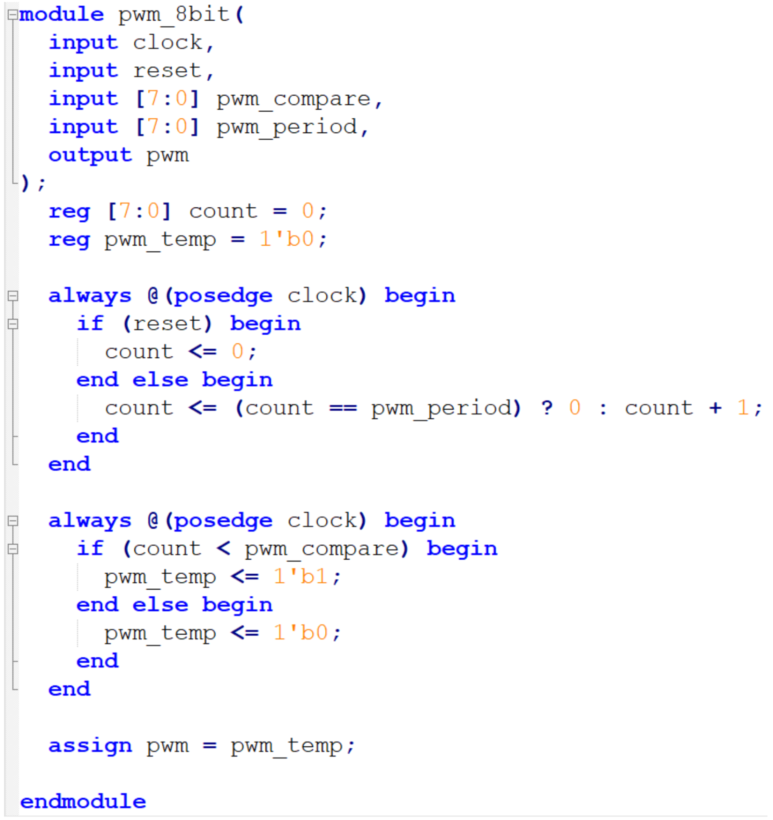


PWM Bits = 8bit

PWM Period = 0-255

PWM Compare = 0-255

https://en.wikipedia.org/wiki/Pulse-width_modulation

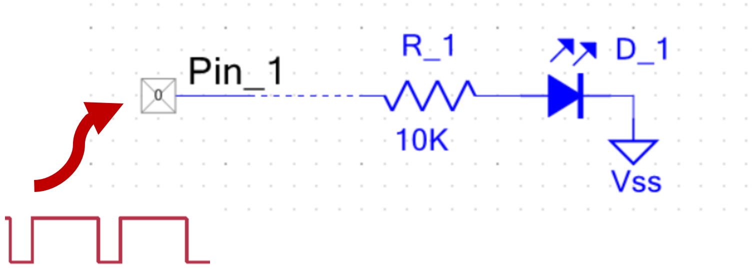


```
module pwm_8bit(
   input clock,
   input reset,
   input [7:0] pwm_compare,
   input [7:0] pwm_period,
   output pwm
);
   reg [7:0] count = 0;
   reg pwm_temp = 1'b0;

   always @(posedge clock) begin
      if (reset) begin
         count <= 0;
      end else begin
         count <= (count == pwm_period) ? 0 : count + 1;
      end
   end

   always @(posedge clock) begin
      if (count < pwm_compare) begin
         pwm_temp <= 1'b1;
      end else begin
         pwm_temp <= 1'b0;
      end
   end

   assign pwm = pwm_temp;

endmodule
```

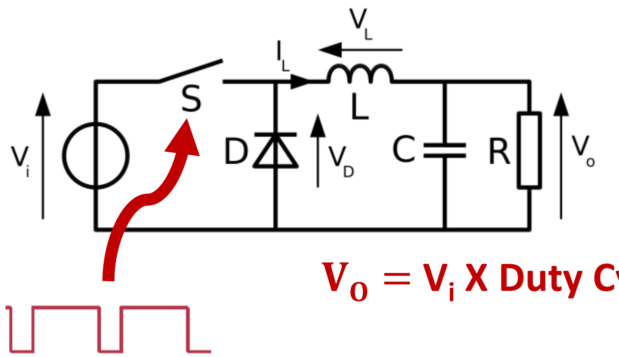
Prof. Kurian Polachan, IIIT-Bangalore

Applications

a. LED Brightness Ctrl



b. SMPS



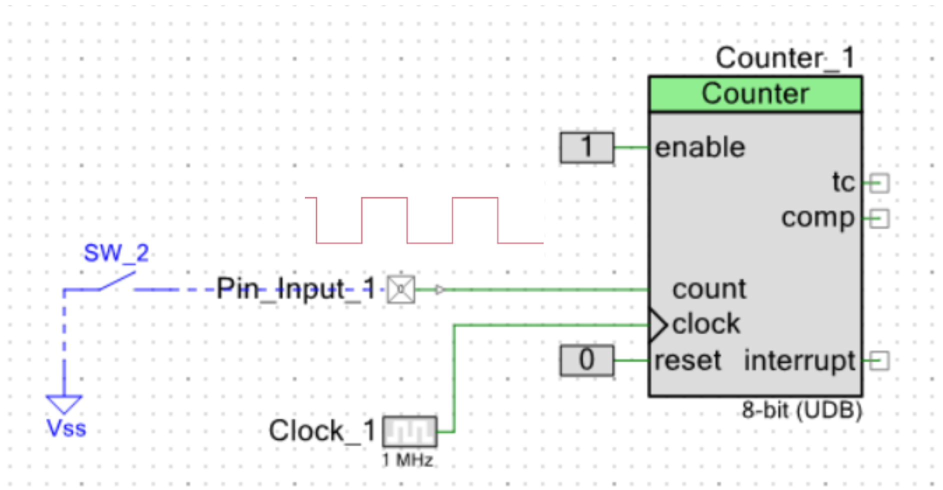
$$V_O = V_i \text{ X Duty Cycle}$$

https://en.wikipedia.org/wiki/Pulse-width_modulation

Counter

Counter

"Used for Counting Events"



Bits = 8bit, 16bit …

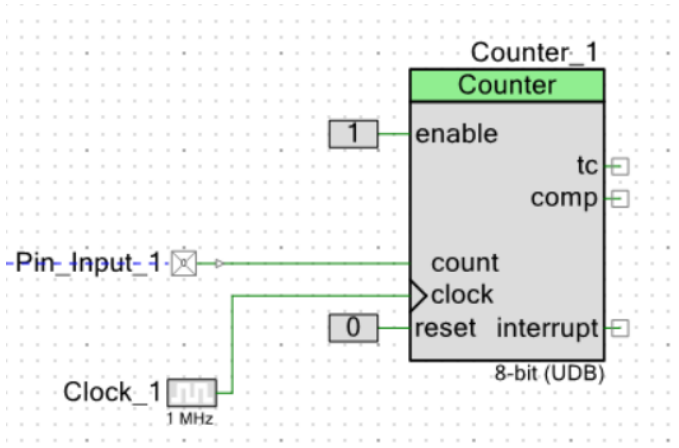
Period ~ $0 - 2^{\text{Bits}}$

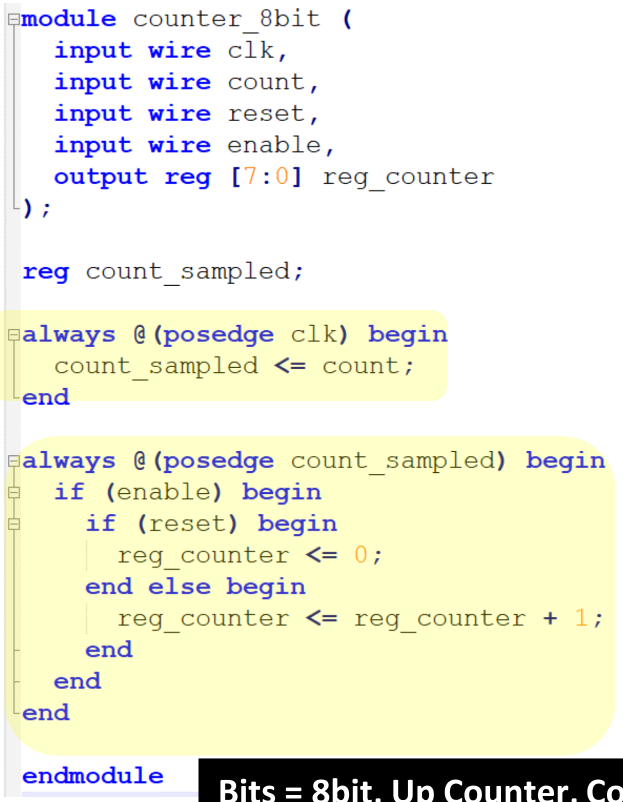
Direction = Up or Down

Count-Edge = Fall or Rise

- The block increments the counter value upon detecting edges (rising or falling) on its count input
- Clock is used for sampling inputs to the counter (to avoid setup violation) > 2x (Signal Freq at Count)
- 'tc' goes high when counter value equals the period (or terminal count). It stays high for one clock cycle

Counter - Verilog





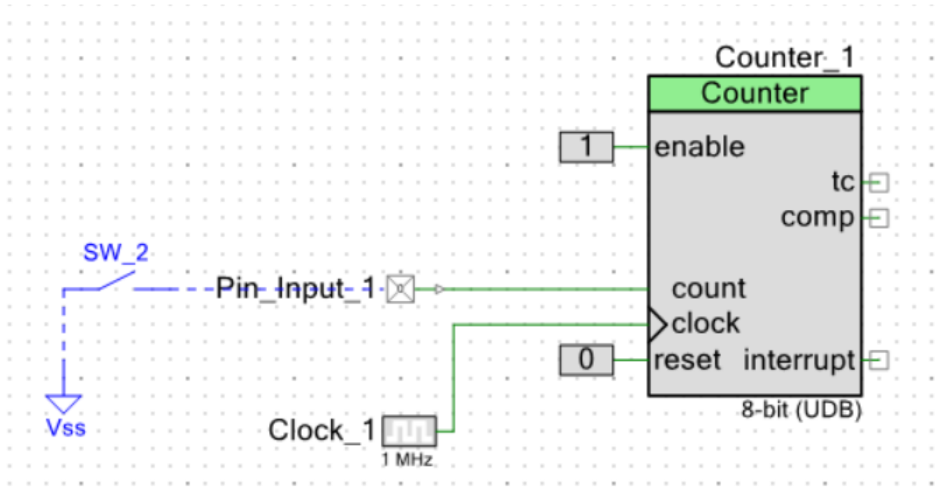
```
module counter_8bit (
   input wire clk,
   input wire count,
   input wire reset,
   input wire enable,
   output reg [7:0] reg_counter
);

reg count_sampled;

always @(posedge clk) begin
   count_sampled <= count;
end

always @(posedge count_sampled) begin
   if (enable) begin
      if (reset) begin
         reg_counter <= 0;
      end else begin
         reg_counter <= reg_counter + 1;
      end
   end
end

endmodule
```

Bits = 8bit, Up Counter, Count-Edge ?

Reading

Why you need clock ? To synchronize asynchronous inputs such as “Count”

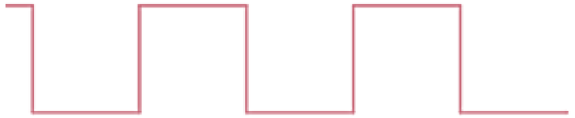
https://web.stanford.edu/class/ee183/handouts/synchronization_pres.pdf

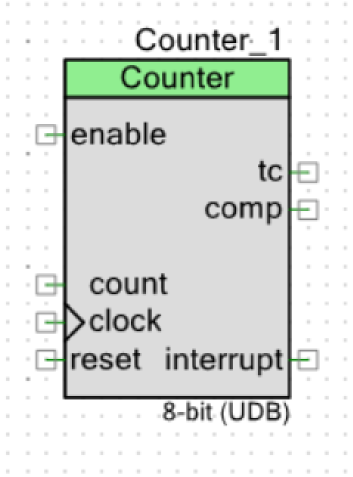


https://en.wikipedia.org/wiki/Pulse-width_modulation

E.g., Detect Signal Freq

"Detect Frequency of a Unknown PWM Signal"





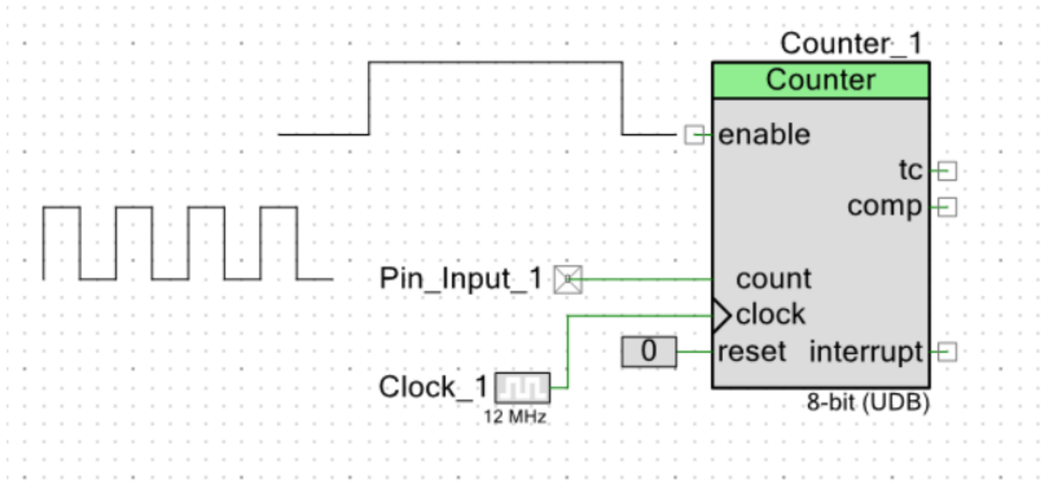
?

- The block increments the counter value upon detecting edges (rising or falling) on its count input
- Clock is used for sampling inputs to the counter (to avoid setup violation) > 2x (Signal Freq at Count)

https://en.wikipedia.org/wiki/Pulse-width_modulation
Ref – creative commons

E.g., Detect Sig Freq (…)

“Detect Frequency of a Unknown PWM Signal”



Enable Time ?

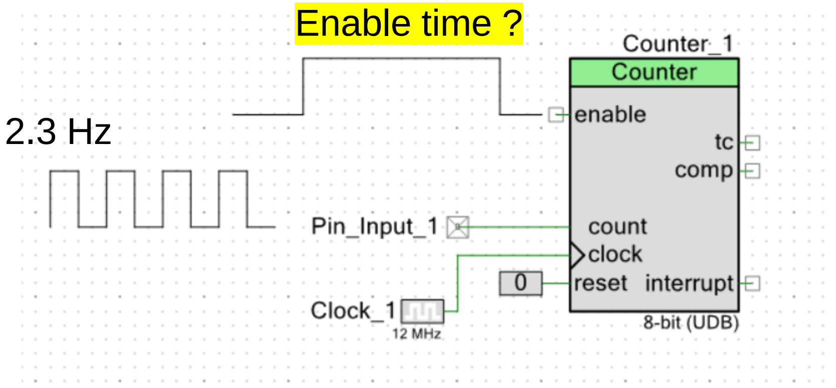
Period ?

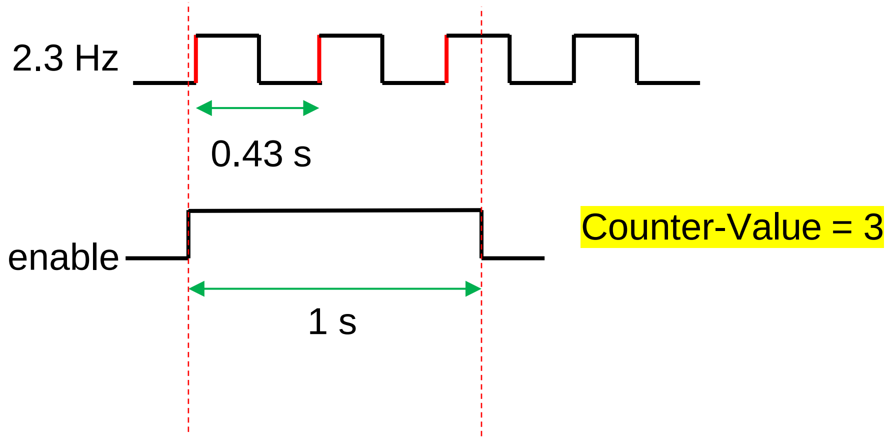
- The block increments the counter value upon detecting edges (rising or falling) on its count input
- Clock is used for sampling inputs to the counter (to avoid setup violation) > 2x (Signal Freq at Count)

https://en.wikipedia.org/wiki/Pulse-width_modulation

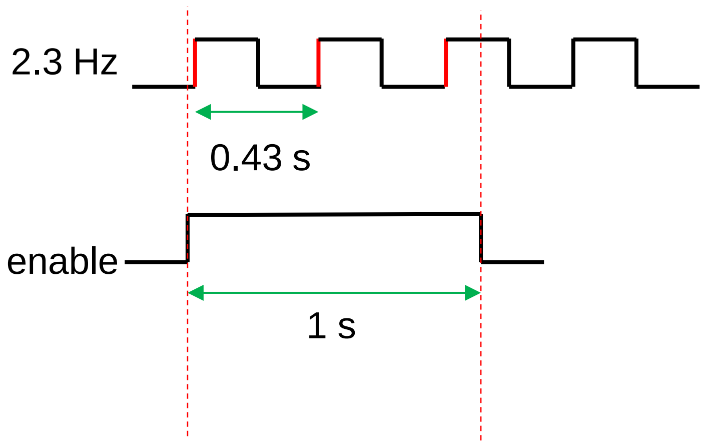
E.g., Detect Sig Freq (…)

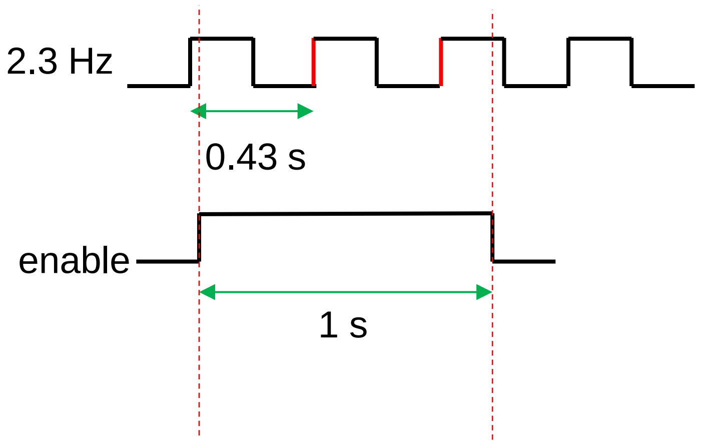
“Detect Frequency of a Unknown PWM Signal”





E.g., Detect Sig Freq (…)





Counter-Value = 3 (~ 3Hz)

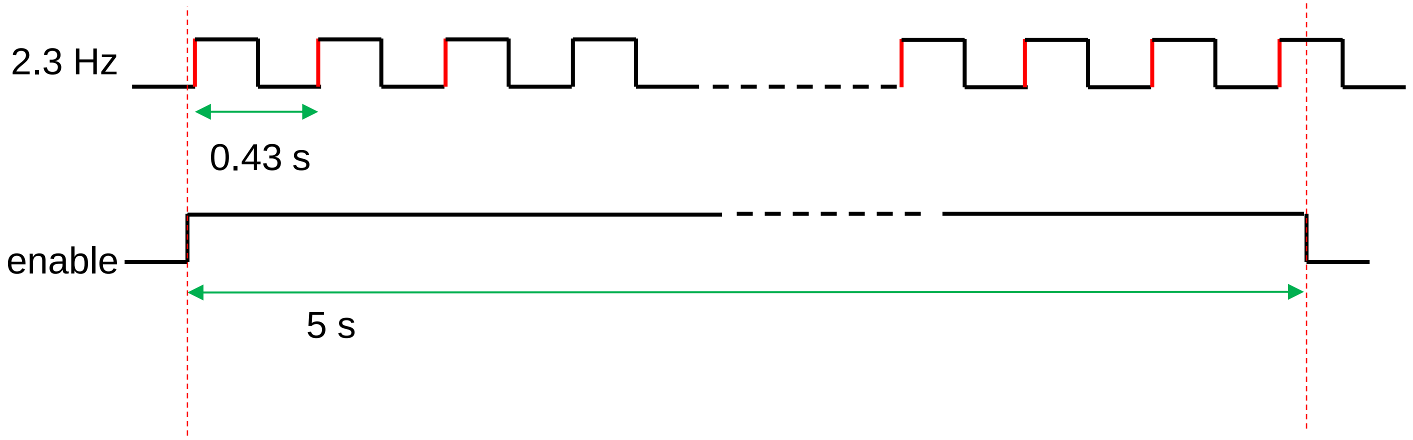
Error % = (3-2.3)/2.3 = 30%

Counter-Value = 2 (~ 2Hz)

Error % = (2-2.3)/2.3 = -13%

Resolution of 1s enable signal is 1Hz. (i.e., counter-output thus could be only 2 or 3)

E.g., Detect Sig Freq (…)

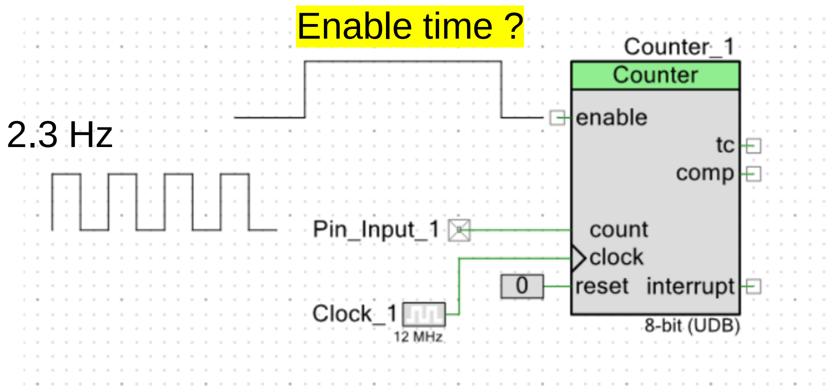


Counter-Value = 11 or 12,   i.e., 2.2 Hz or 2.4 Hz

Error = 100*(2.4-2.3)/2.3 = 4.3%, 100*(2.2-2.3)/2.3 = ?

Note – For enable of 5 seconds, 5 count correspond to 1Hz, i.e., 1 count corresponds to 0.2Hz

E.g., Detect Sig Freq (...)



i.e., in this setting a signal frequency less than 1Hz cannot be detected.

$$\overbrace{\hspace{6cm}}$$

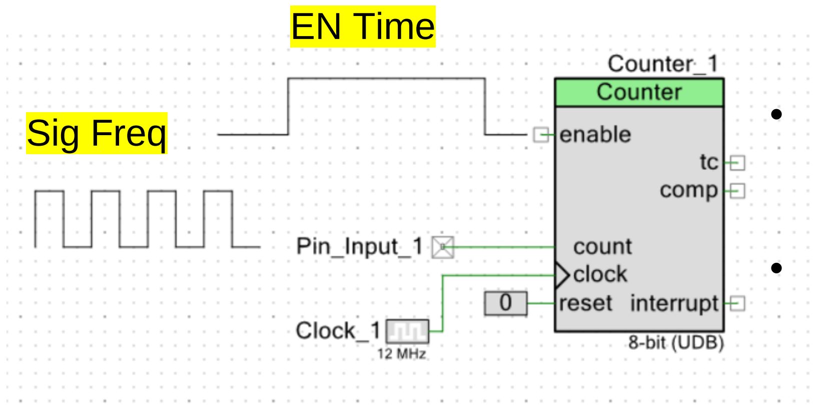
For enable of 1 seconds, 1 count correspond to 1 Hz → Resolution of the counter = 1 Hz

For enable of 5 seconds, 1 count correspond to 0.2 Hz → Resolution = 0.2 Hz

For enable of 10 seconds, 1 count correspond to 0.1 Hz → Resolution = 0.1 Hz

For enable of 100 seconds, 1 count correspond to 0.01 Hz → Resolution = 0.01 Hz

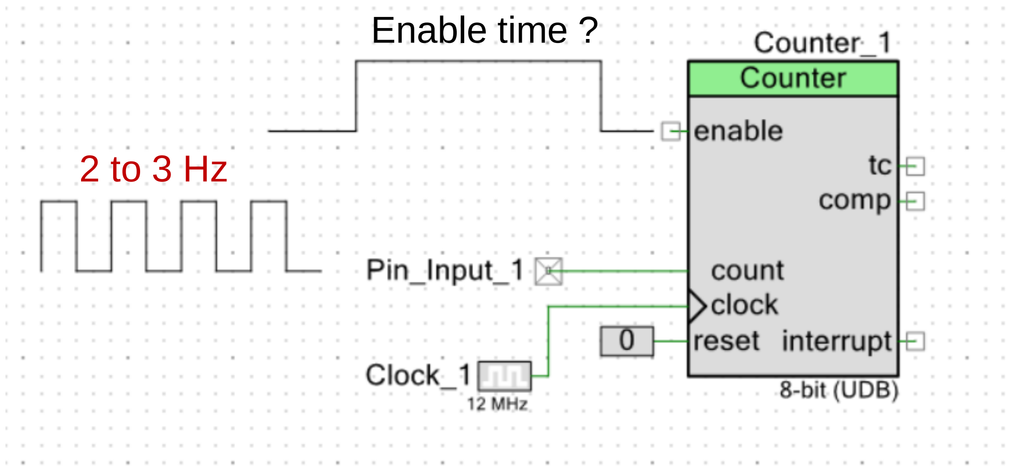
Question



- **EN Time = 1s, Sig Freq = 2.3456Hz, Counter Out ?**
- **EN Time = 1s, Sig Freq = 2.8456Hz, Counter Out ?**
- **EN Time = 10s, Sig Freq = 2.3456Hz, Counter Out ?**
- **EN Time = 100s, Sig Freq = 2.3456Hz, Counter Out ?**

Question ?

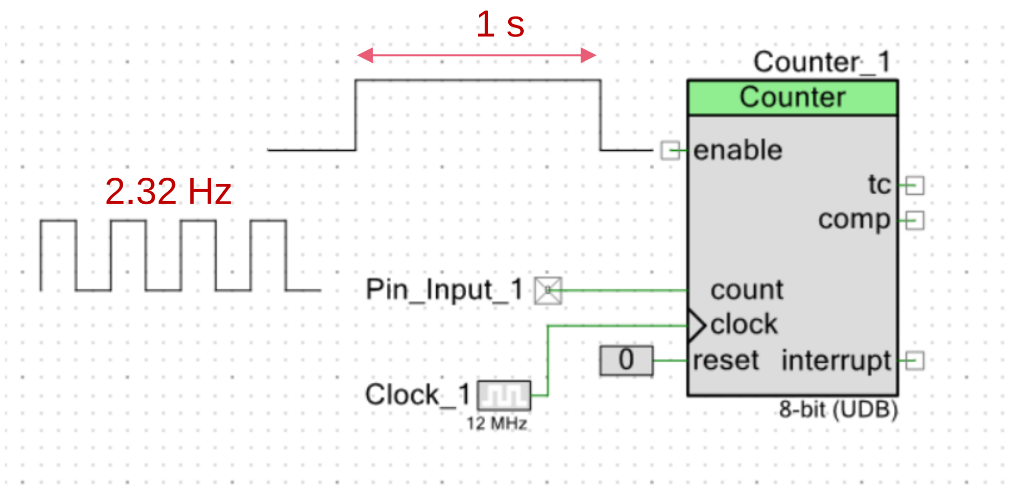
I need a resolution of 0.01 Hz with a 8bit counter to measure a signal with frequency range between 2 to 3 Hz. What to do ?



Resolution of 0.01 Hz require 100seconds of enable time ? A signal frequency in the range of 2 to 3Hz will technically require a counter which can count up to ? (3*100=300)

Solution → Averaging

I need a resolution of 0.01 Hz with a 8bit counter to measure a signal with frequency range between 2 to 3 Hz. What to do ?



Solution: Perform measurements a large number of time.

- Counter Outputs = 2 or 3
- 2 for 68% and 3 for 32%
- **Averaging → 2.32**

# of Measurements Required ?

Prof. Kurian Polachan, IIIT-Bangalore