EMBEDDED

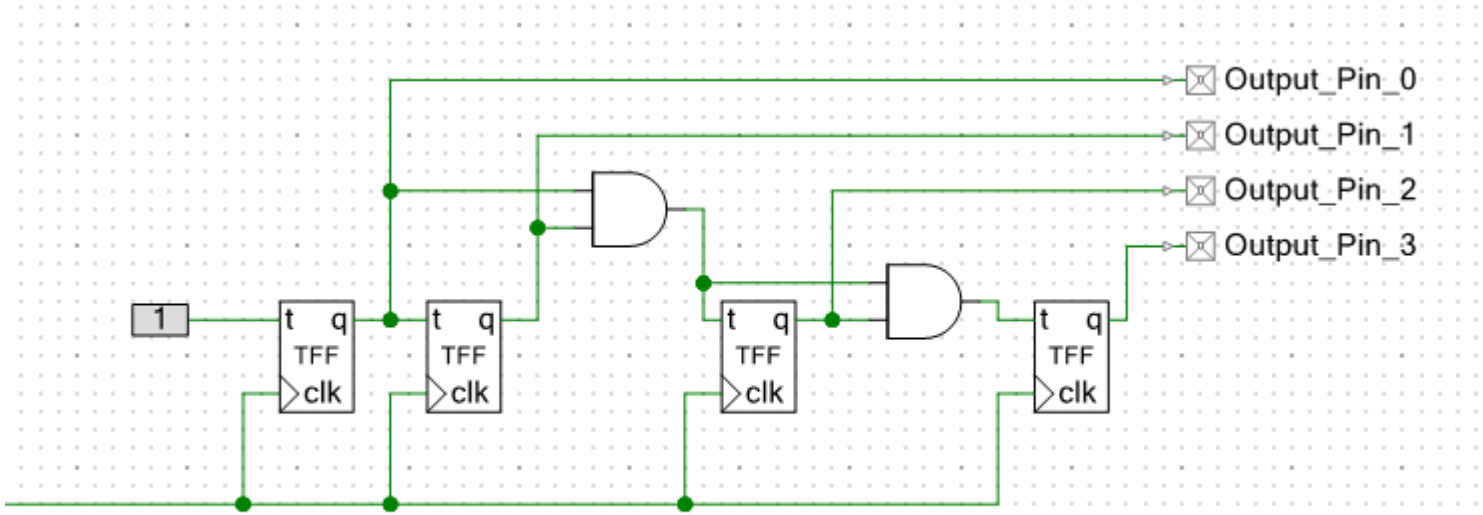
л.6
Лічильник, Таймер, ШІМ

Палій Святослав

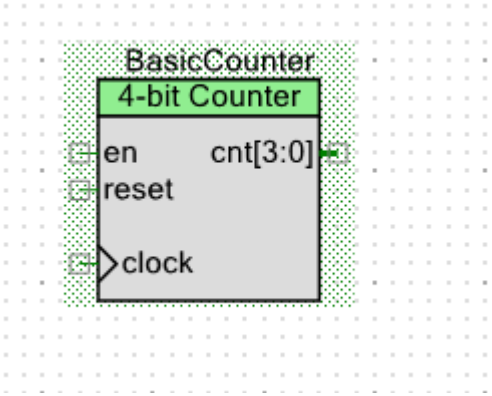
Український Католицький Університет
APPLIED SCIENCES FACULTY

Ми повинні вміти «комунікувати» з часом

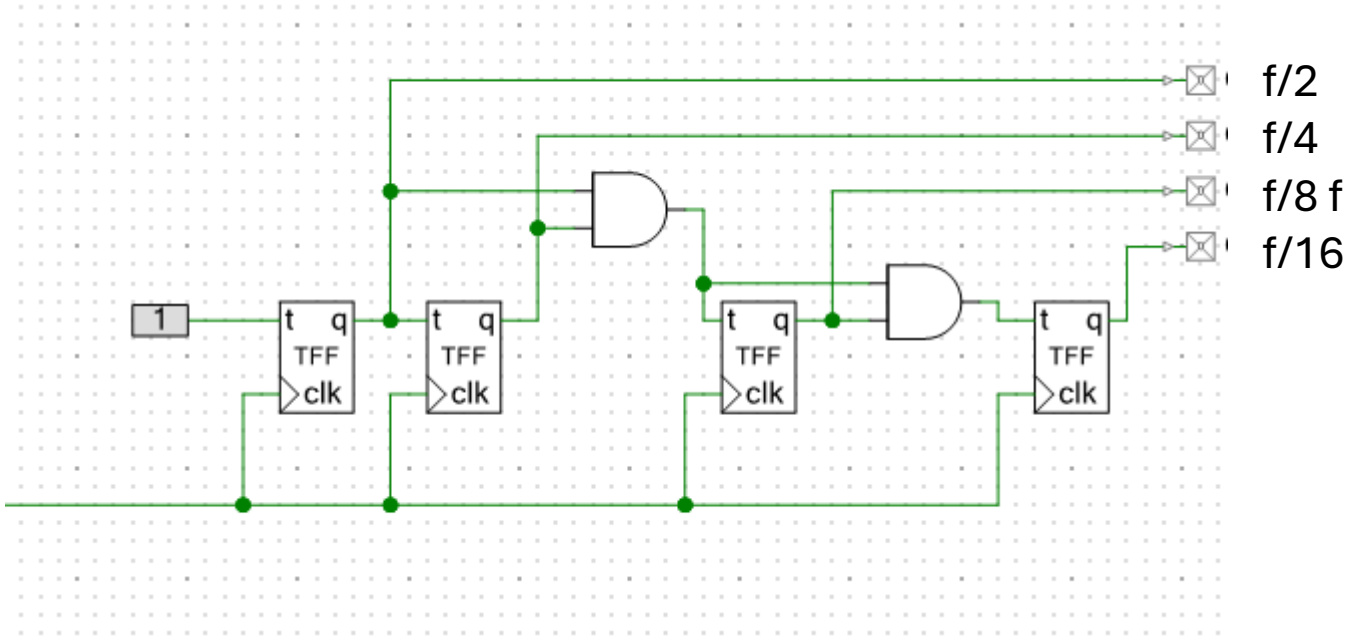
Ми нещодавно розглядали побудову лічильника в процесорі PSoC4



- Кожен імпульс тактового входу лічильник збільшує (або зменшує) свій рахунок на одиницю
- Лічильник можна побудувати і на інших видах тригерів. D-тригенах, JK-тригерах, тощо



Лічильник як подільник частоти

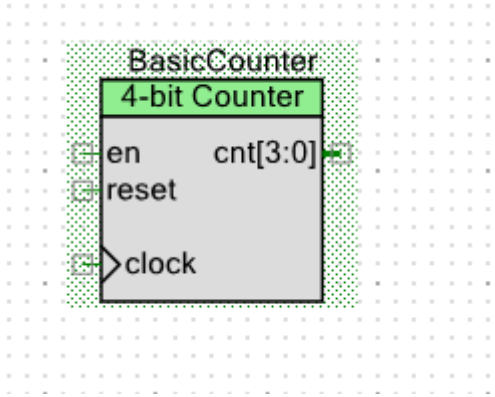


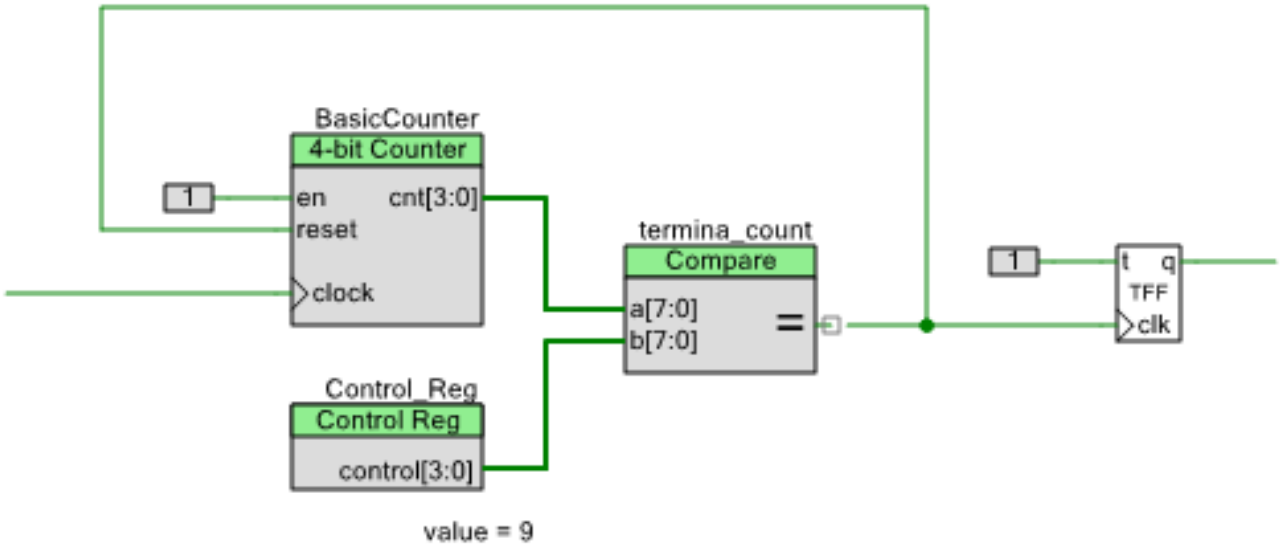
Лічильник можна використовувати як подільник частоти на число кратне 2^X.
Цим справді часто користуються

Image from Wikipedia

Найбільш базову схему варто трохи доробити

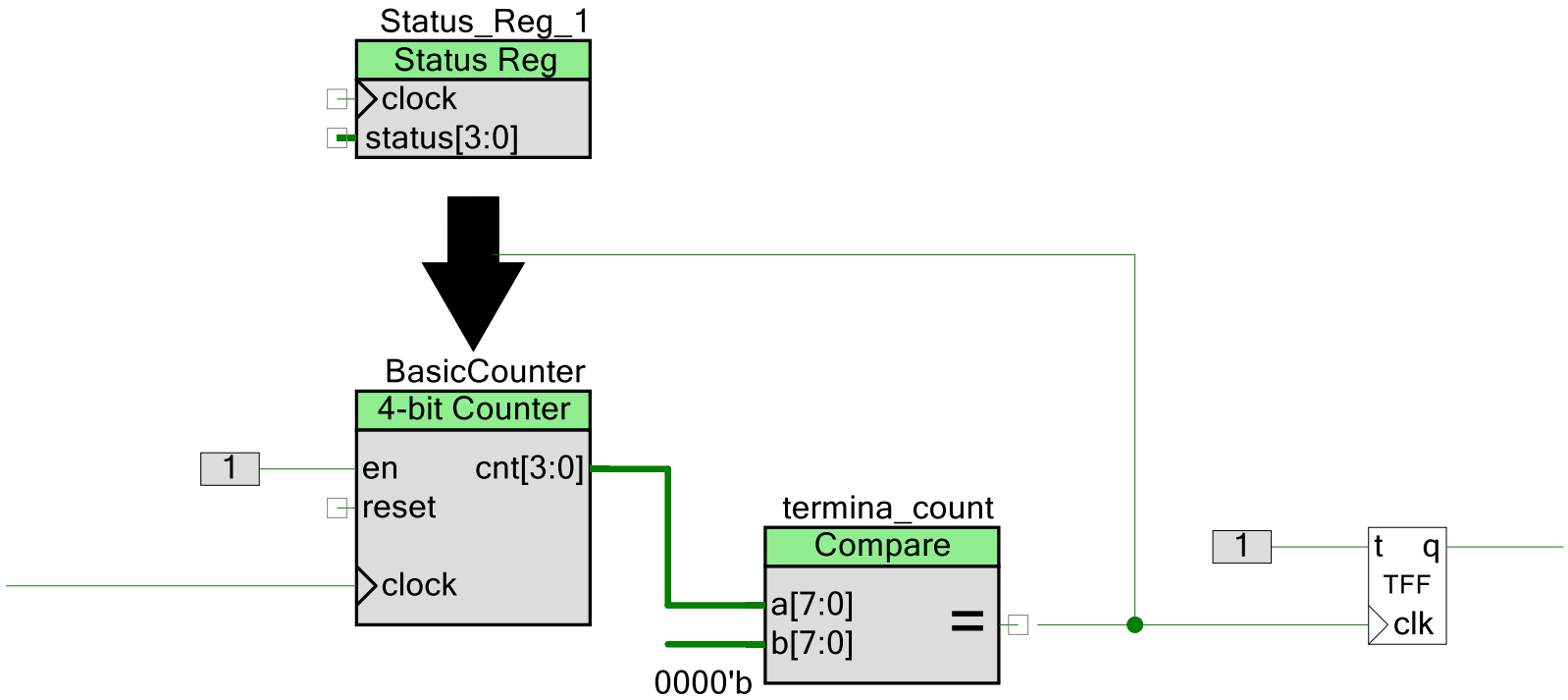
- Додати вхід дозволу рахунку
- Додати вхід скиду у початковий стан

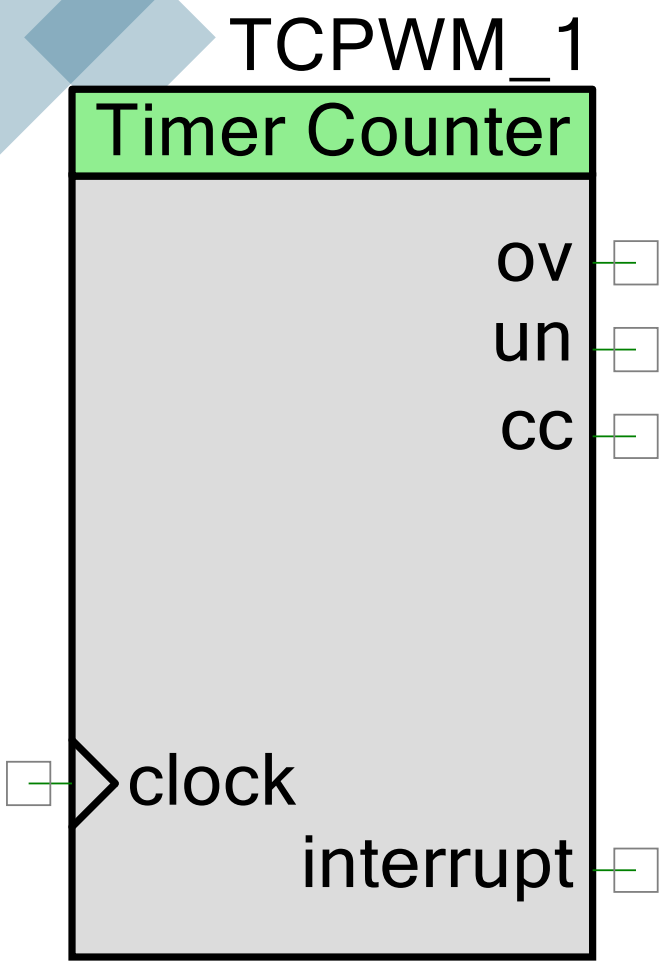


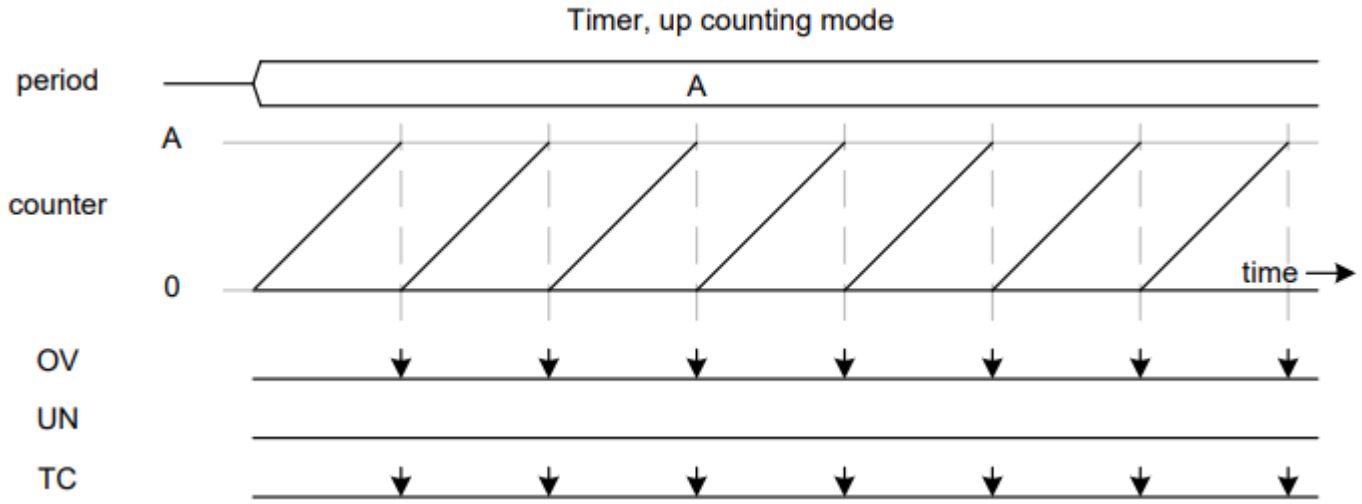


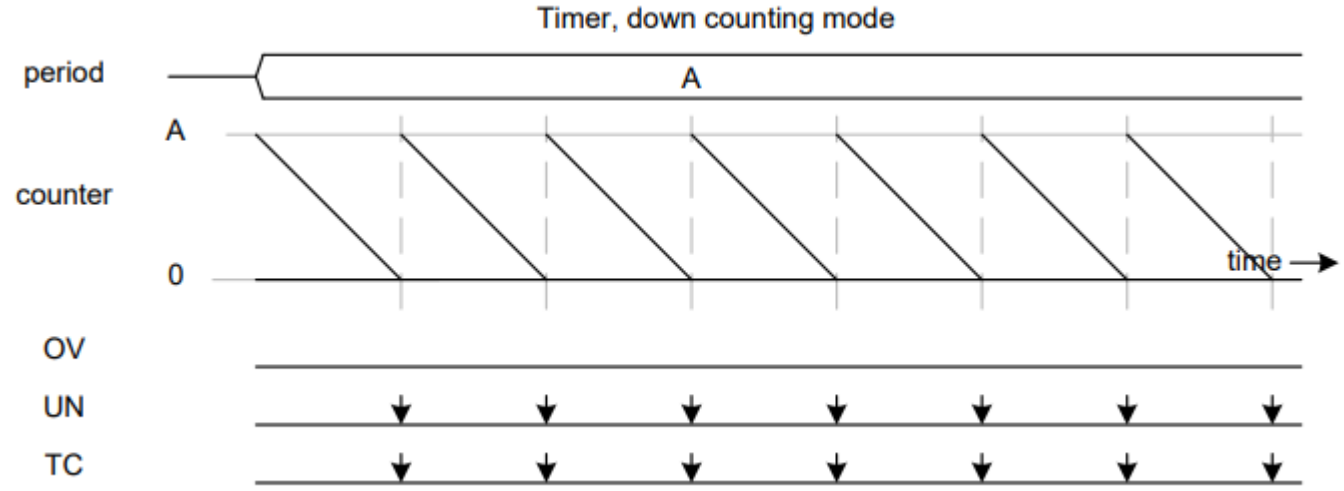
Такий лічильник може вже ділити частоту на довільне число в межах його розрядності

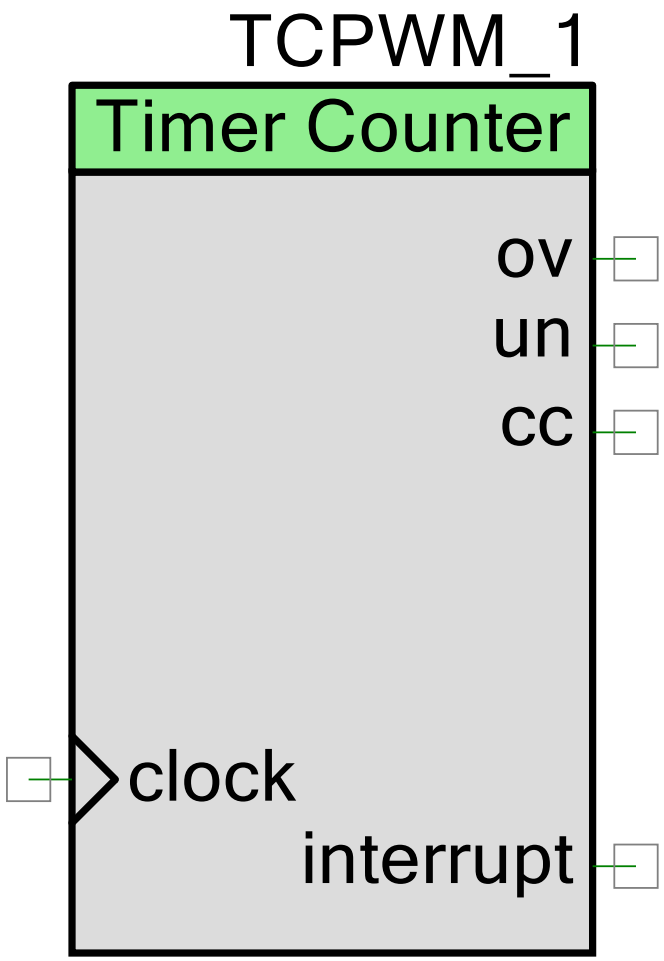
Для лічильника котрий рахує вниз може бути замість схеми котра скидає в 0 мати схему котра дозволяє завантажити значення

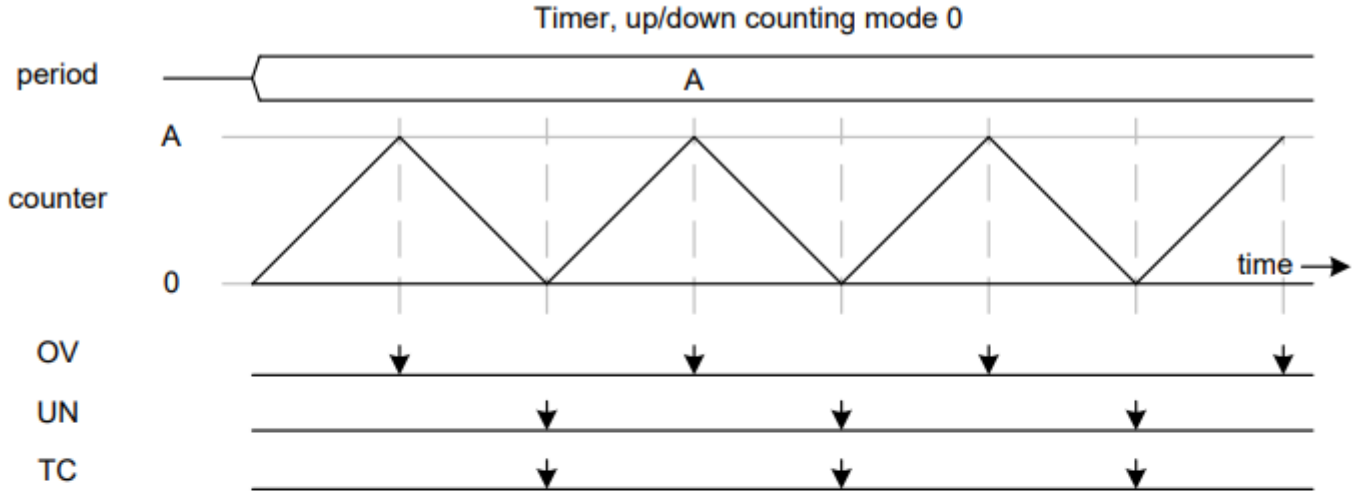


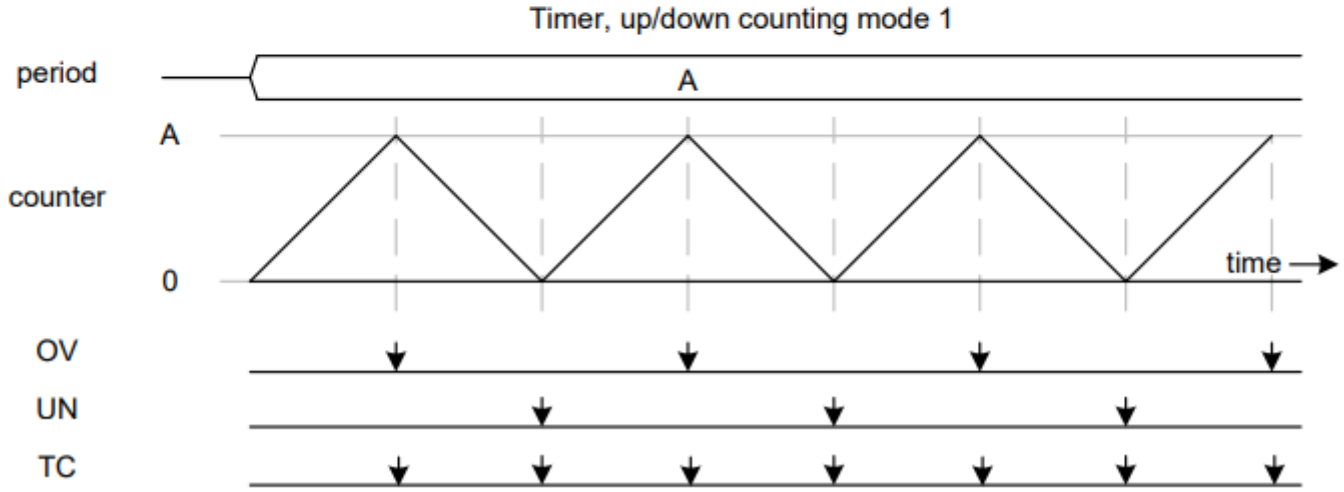

TCPWM_1
Timer Counter
ov
un
cc
clock
interrupt

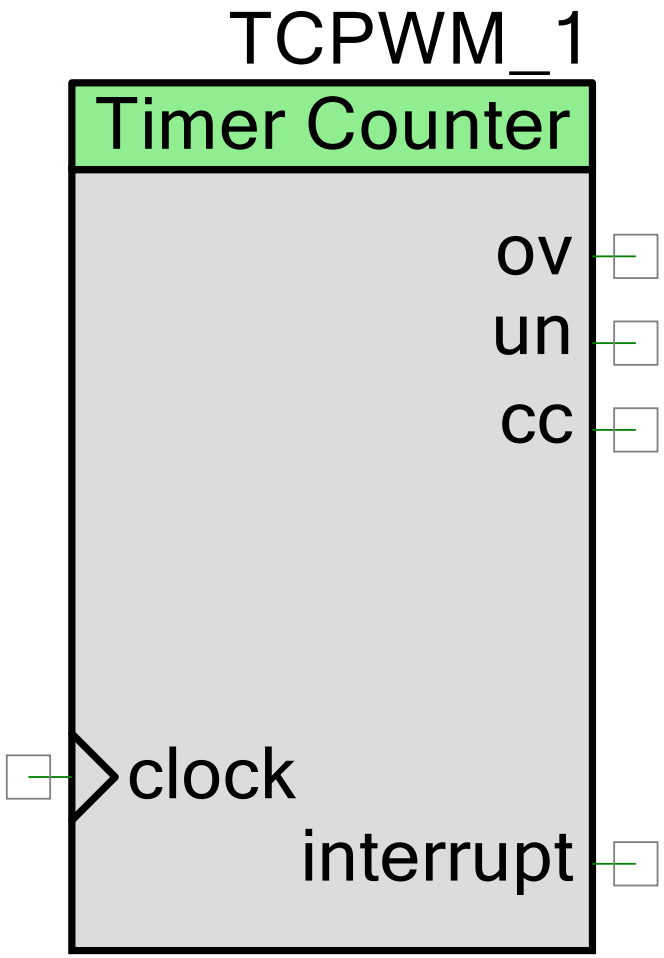

Timer, up counting mode
period
A
A
counter
0
time
OV
UN
TC

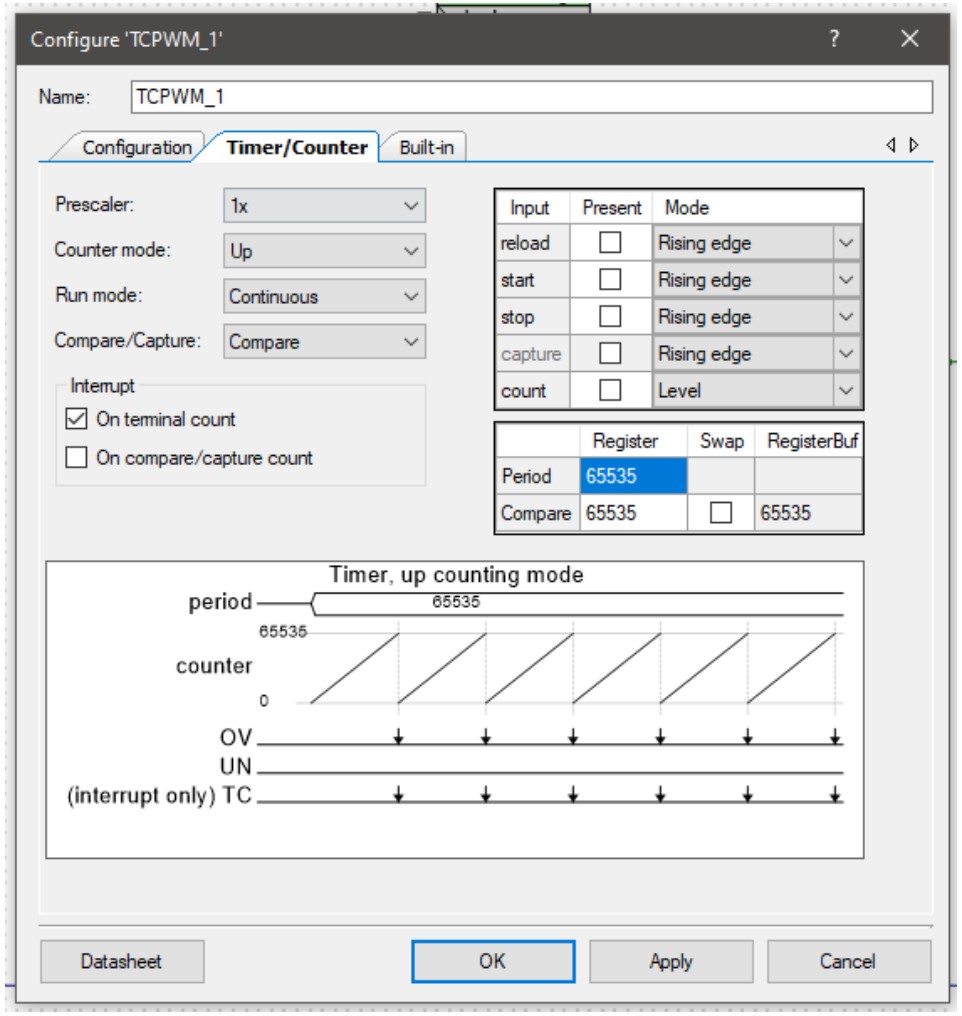

Timer, down counting mode
period
A
A
counter
0
time
OV
UN
TC


TCPWM_1
Timer Counter
ov
un
cc
clock
interrupt


Timer, up/down counting mode 0
period
A
A
counter
0
time
OV
UN
TC


Timer, up/down counting mode 1
period
A
A
counter
0
time
OV
UN
TC





- **OV – OverFlow**
- **UN – UnderFlow**
- **CC – Compare Count**

Програмно доступні функції

Звичайно це все немає змісту без програмного доступу:
Зазвичай дані лічильника доступні у вигляді регістрів в зоні регістрів вводу/виводу (тут пригадуємо що всі вони volatile)

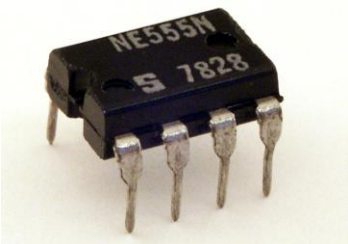
- Значення лічильника
- Регістр завантаження (period value)

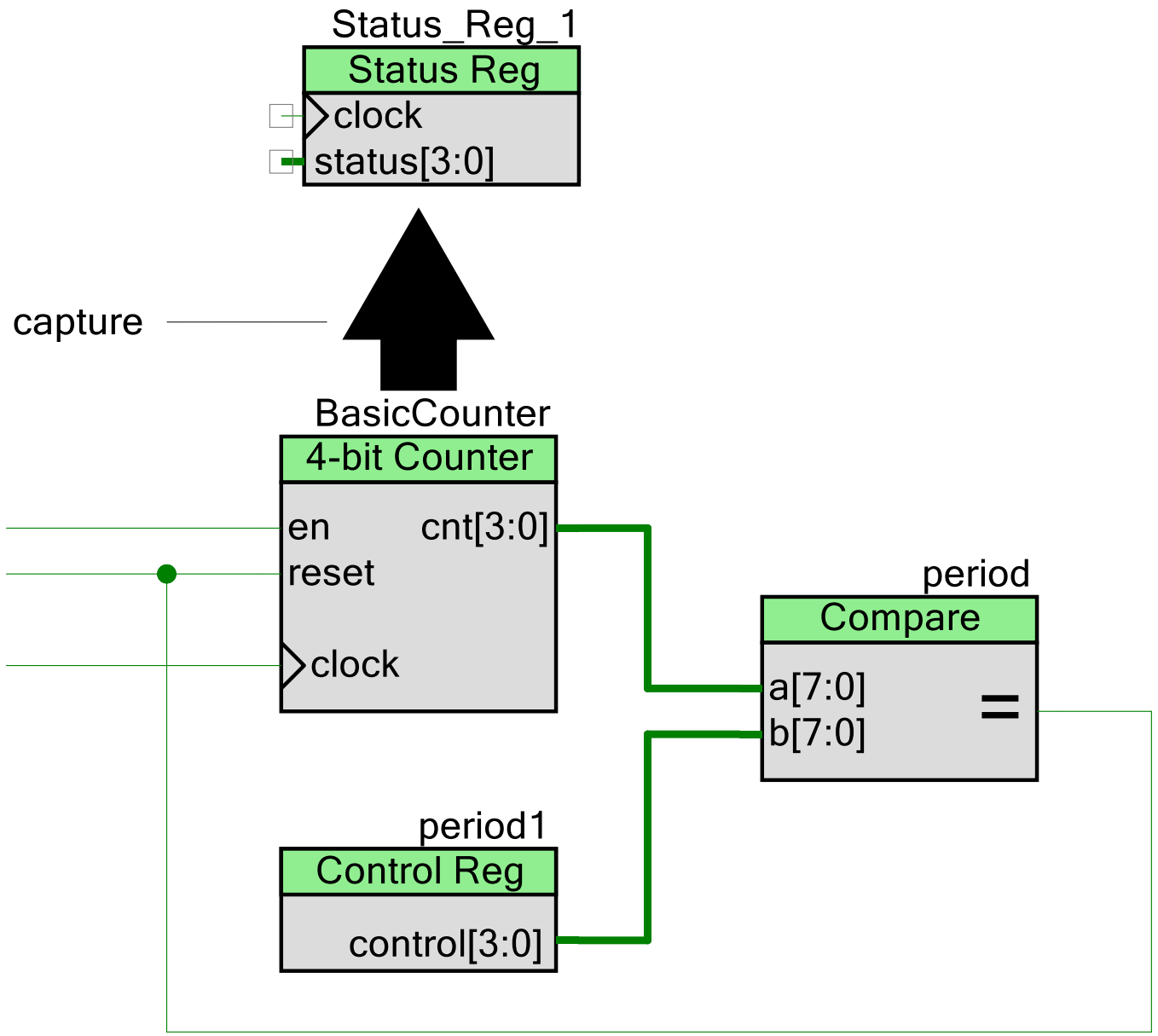
Як правило бібліотеки функцій від виробника мікроконтролера також мають готові підпрограми для взаємодії з лічильником

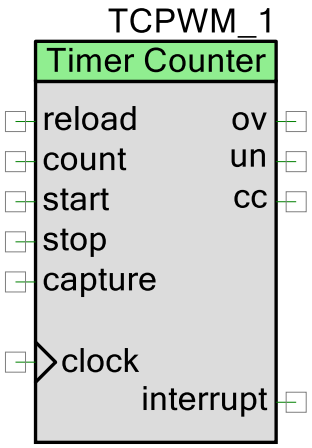
TCPWM_WriteCounter()	Writes a new 16 bit counter value directly into the counter register
TCPWM_ReadCounter()	Reads the current counter value
TCPWM_SetCounterMode()	Sets the counter mode
TCPWM_WritePeriod()	Writes the 16 bit period register with the new period value
TCPWM_ReadPeriod()	Reads the 16 bit period register

А давайте зробимо його таймером

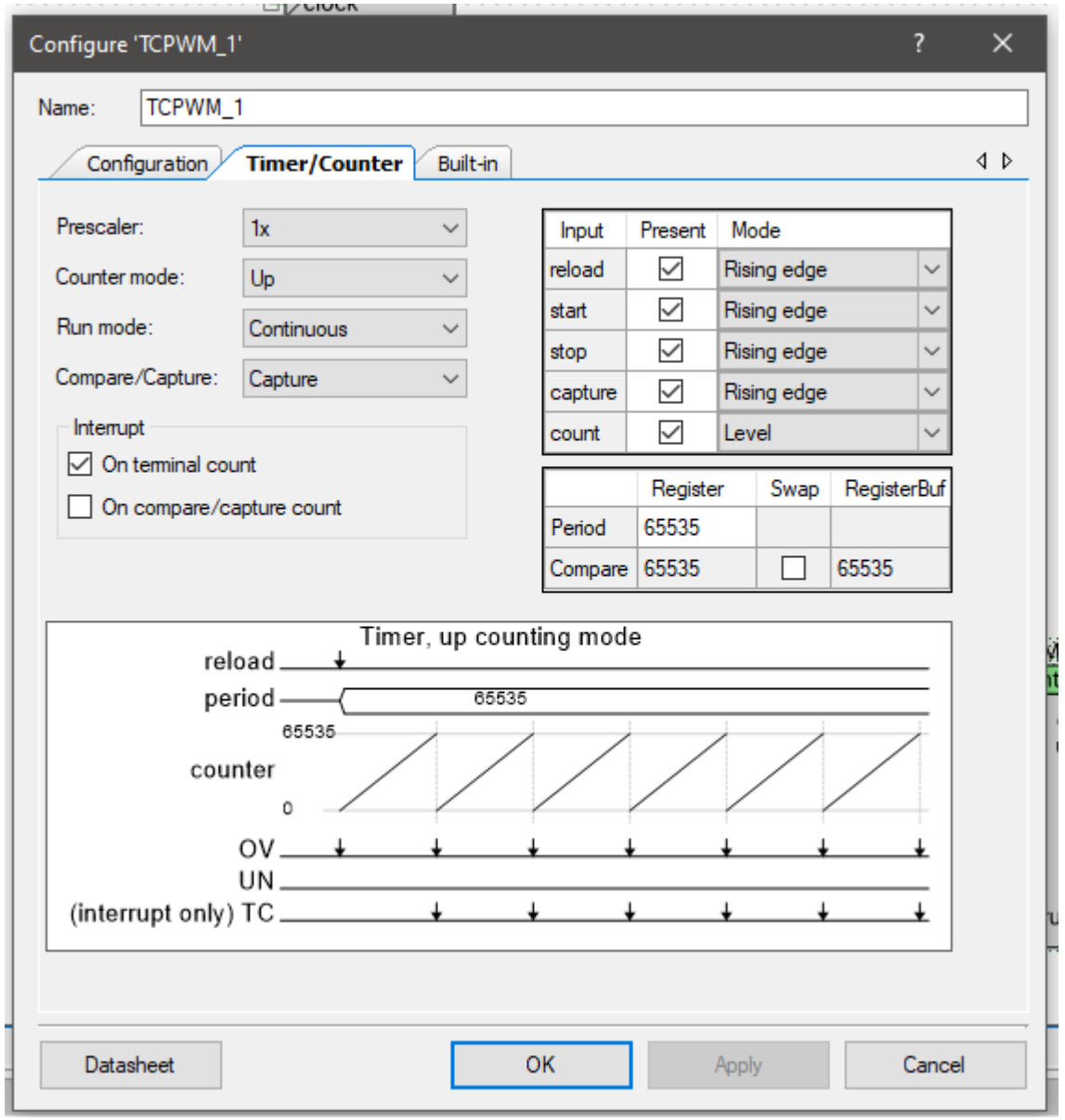
- Насправді різниця між лічильником та таймером є доволі умовна
- Але все ж таки для універсальної та ефективної роботи йому варто додати деякий додатковий функціонал, зокрема додатковий регістр з значенням котре можна порівнювати з значенням лічильника, додатковий регістр котрий може зберігати значення лічильника «зловлене» за певних умов (capture)
- А також певну кількість вхідних сигналів котрі дозволяють його зупиняти, запускати, копіювати вміст лічильника до регістру (capture).
- Більшість функцій котрі ми щойно назвали входами також повинні бути доступними програмно. Тобто можна запустити, зупинити, копіювати лічильник в capture за допомогою запису в регістри вводу/виводу або виклику підпрограм.
- Також вихідні сигнали дозволяють генерувати переривання та встановлюють відповідні біти в програмно доступних регістрах статусу.




Status_Reg_1
Status Reg
clock
status[3:0]
capture
BasicCounter
4-bit Counter
en
reset
clock
cnt[3:0]
period
Compare
a[7:0]
b[7:0]
=
period1
Control Reg
control[3:0]



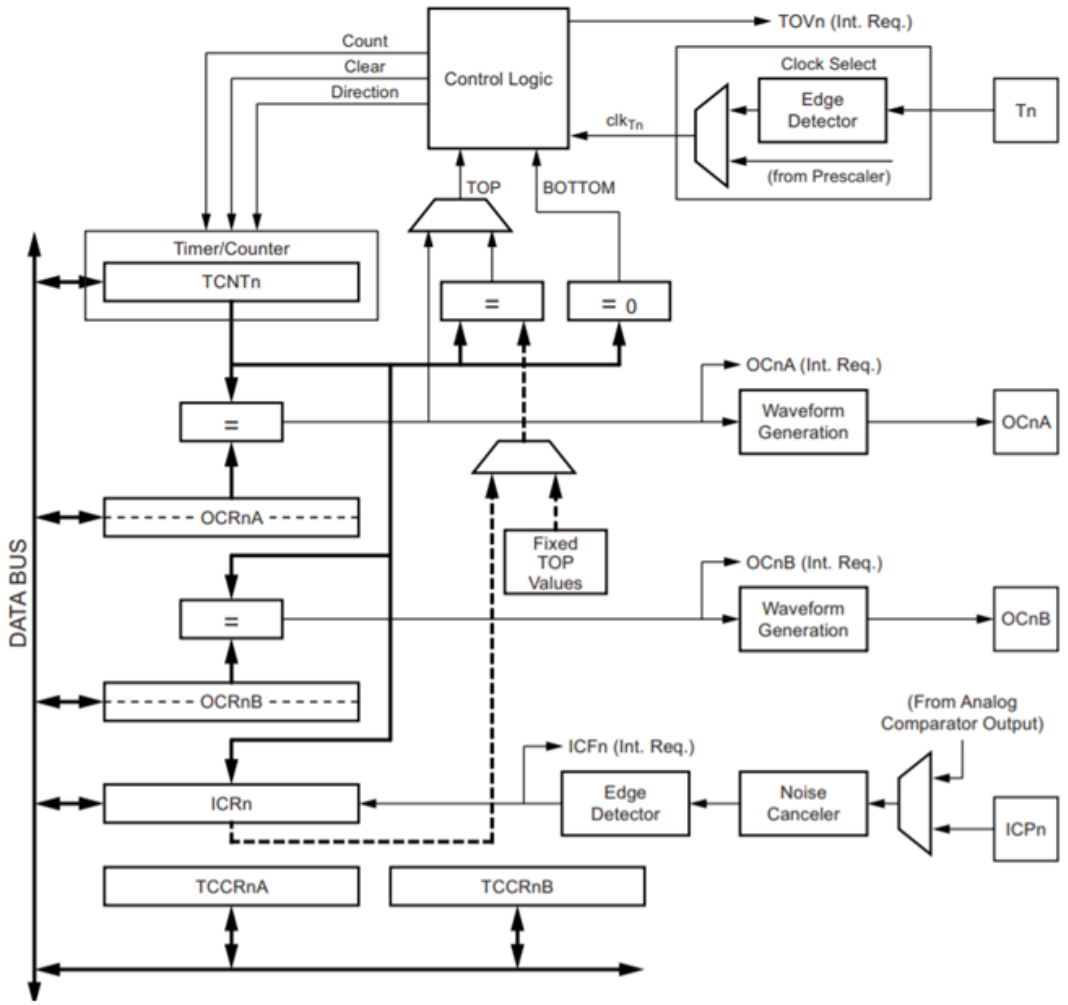
Reload – оновити значення лічильника з регістру period
Start – запустити лічильник
Stop – зупинити лічильник
Capture – копіювати значення лічильника в регістр capture



ATmega328P Timer

16-bit Timer/Counter1

Figure 15-1. 16-bit Timer/Counter Block Diagram(1)



```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile int pulse_count = TCNT1;

void setup() {
  // Встановити пін 2 як вхід
  pinMode(2, INPUT);

  // Заборонити переривання (критична секція)
  cli();

  // Встановити Timer1 в нормальний режим
  TCCR1A = 0;

  // Налаштувати інкремент Timer1 про зростаючому фронту пін 2
  TCCR1B = (1 << CS12) | (1 << CS11) | (1 << CS10) | (1 << ICES1);

  // обнулити лічильник таймера
  TCNT1 = 0;

  // Дозволити переривання від таймера
  TIMSK1 = (1 << ICIE1);

  // Enable global interrupts
  sei();
}

ISR(TIMER1_CAPT_vect) {
  // Це переривання виникає кожен раз при зростаючому фронті на пін 2
  // Читаємо значення лічильника
  pulse_count = TCNT1;
}
```

Приклад:
рахуємо імпульси

```c
#include <avr/io.h>
#include <avr/interrupt.h>

volatile uint16_t pulse_length = 0;

void setup() {
  pinMode(8, INPUT);

  // заборона переривання
  cli();

  // нормальний режим таймера
  TCCR1A = 0;

  // налаштувати запуск таймера по сигналу від pin 8
  TCCR1B = (1 << CS12) | (1 << CS11) | (1 << CS10) | (1 << ICES1);

  // очистити лічильник таймера
  TCNT1 = 0;

  // дозволити переривання capture (від pin8 під'єднаного до лічильника)
  TIMSK1 = (1 << ICIE1);

  // дозволити переривання
  sei();
}

ISR(TIMER1_CAPT_vect) {
  // зберегти довжину імпульса
  pulse_length = ICR1;

  // обнулити лічильник
  TCNT1 = 0;
}
```

Приклад:
рахуємо тривалість імпульса

В яких одиницях буде тривалість?

Тривалість буде виміряна в кількості тактових імпульсів на вході таймера. Виходячи з частоти тактового сигналу ми можемо визначити тривалість одного тактового імпульсу = 1/f а отже і тривалість виміряного сигналу.

```
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>

void setup() {
  // налаштувати на період 1 сек та дозволити переривання від Watchdog Timer
  wdt_enable(WDTO_1S);
  WDTCSR |= (1 << WDIE);

  // Дозволити всі переривання
  sei();
}

ISR(WDT_vect) {
}

void goToSleep() {
  // встановити режим сну
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();

  // заснути
  sleep_cpu();
            // <<<<<   тут чекаємо поки нас розбудить таймер
  // проснутися
  sleep_disable();
}

void loop() {
  Serial.println("Woke up!");

  // Put the microcontroller to sleep
  goToSleep()
}
```

Приклад: будуємо енергоощадний код

PWM

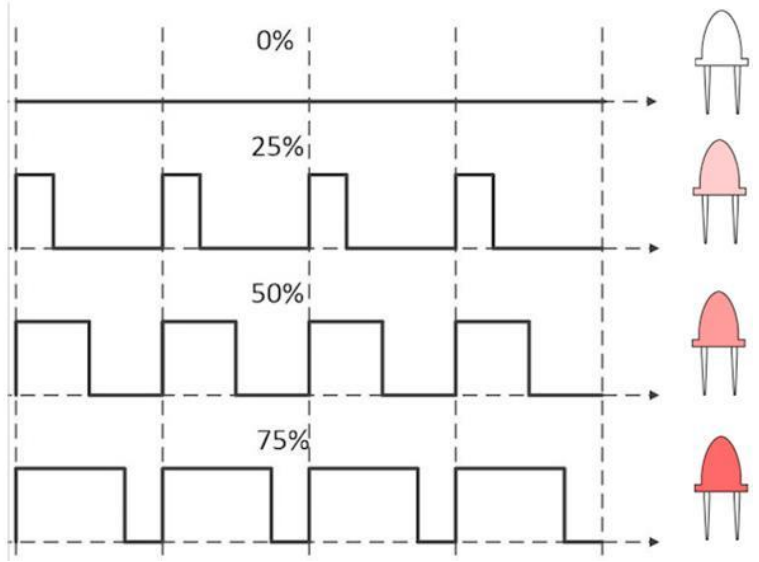
- Якщо до таймера додати ще один компаратор котрий буде порівнювати значення лічильника з значенням з compare register то ми отримаємо ШІМ (PWM).
- ШІМ, або широтно-імпульсна модуляція (англ. PWM, Pulse Width Modulation) — це метод модуляції, який використовується для керування аналоговими сигналами за допомогою цифрових засобів. ШІМ дозволяє регулювати середнє значення напруги чи струму, яке подається на навантаження, шляхом зміни ширини імпульсів.

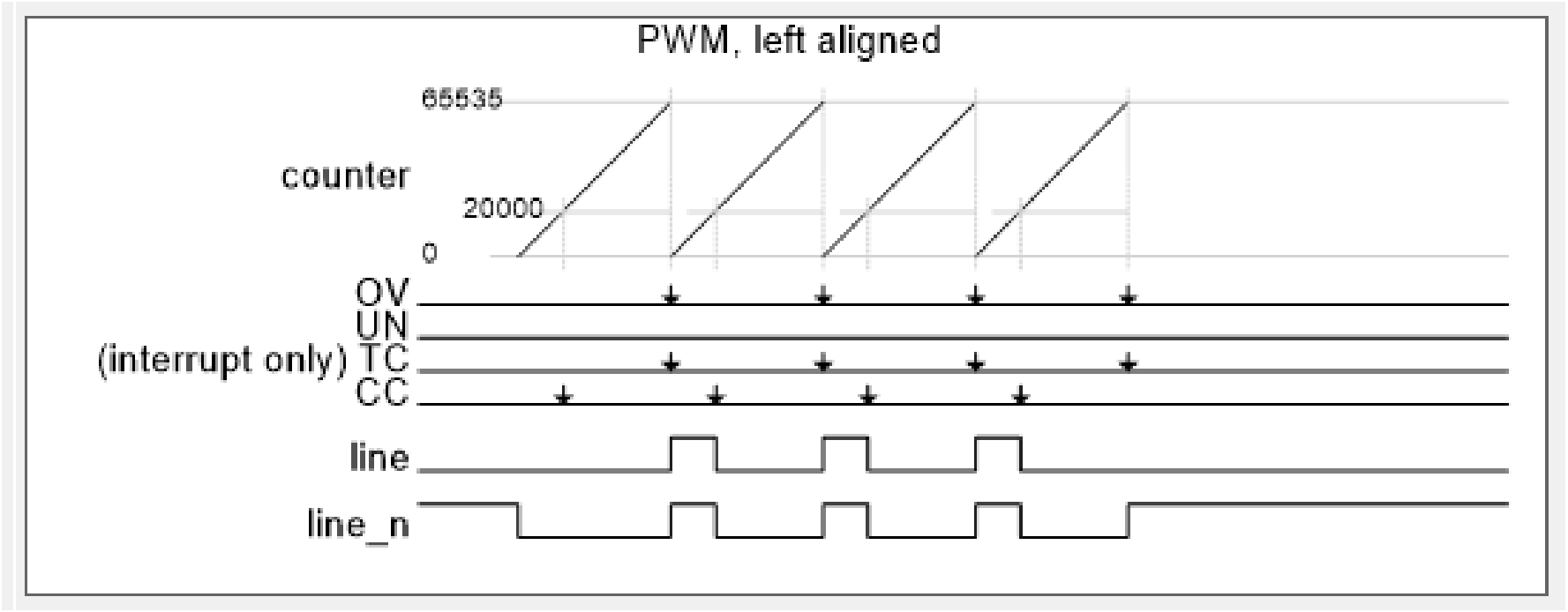
Як працює ШІМ:

1. **Частота**: ШІМ сигнал має певну частоту, яка визначає кількість імпульсів в одиницю часу.
2. **Заповнення (Duty Cycle)**: Це відношення часу, протягом якого сигнал знаходиться у високому стані (включений), до загального періоду сигналу. Вимірюється у відсотках.
    1. **0%**: Сигнал завжди вимкнений.
    2. **50%**: Сигнал включений половину часу і вимкнений іншу половину.
    3. **100%**: Сигнал завжди включений.

Застосування ШІМ:

- **Керування двигунами**: Регулювання швидкості обертання двигунів постійного струму.
- **Регулювання яскравості світлодіодів**: Зміна яскравості світлодіодів шляхом регулювання середнього значення напруги.
- **Аудіо**: Генерація звукових сигналів та керування гучністю.
- **Живлення**: Регулювання напруги живлення в імпульсних джерелах живлення.





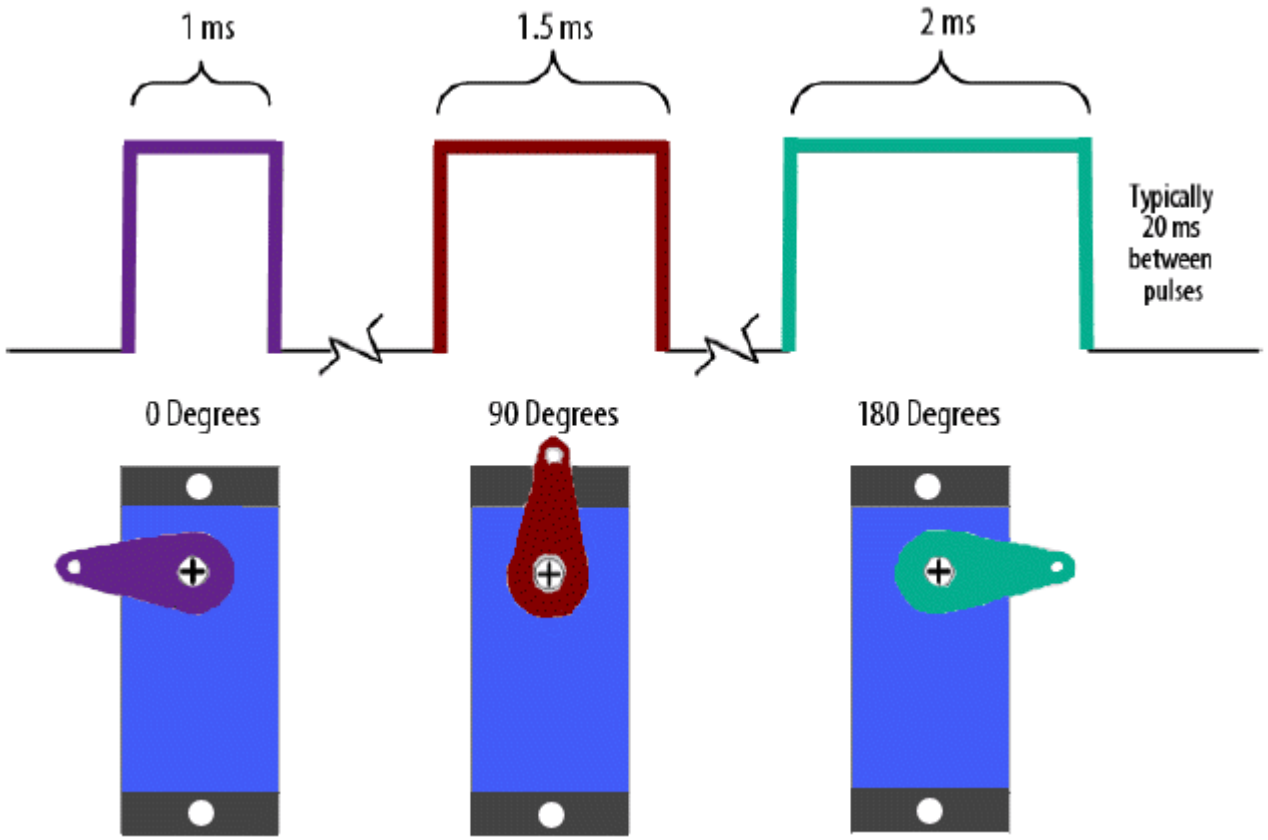
	Register
Period	65535
Compare	20000

Приклад:

LED плавно спалахує та погасає

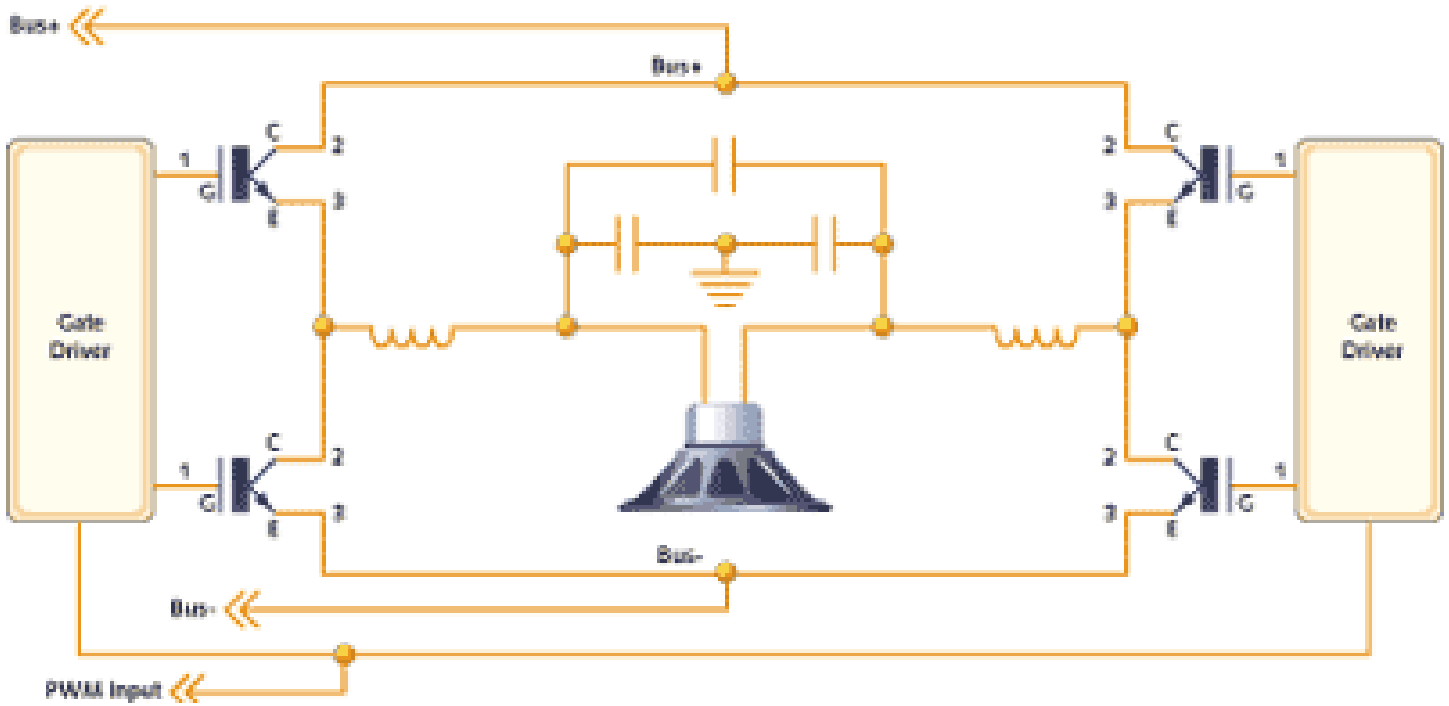
```
int ledPin = 9; // Пін для підключення світлодіода

void setup() {
  pinMode(ledPin, OUTPUT); // Встановлюємо пін як вихід
}

void loop() {
  for (int brightness = 0; brightness <= 255; brightness++) {
    // Встановлюємо яскравість
    analogWrite(ledPin, brightness);

    // Затримка для плавного зміни яскравості
    delay(10);
  }

  for (int brightness = 255; brightness >= 0; brightness--) {
    // Встановлюємо яскравість
    analogWrite(ledPin, brightness);

    // Затримка для плавного зміни яскравості
    delay(10);
  }
}
```

Керування сервоприводами



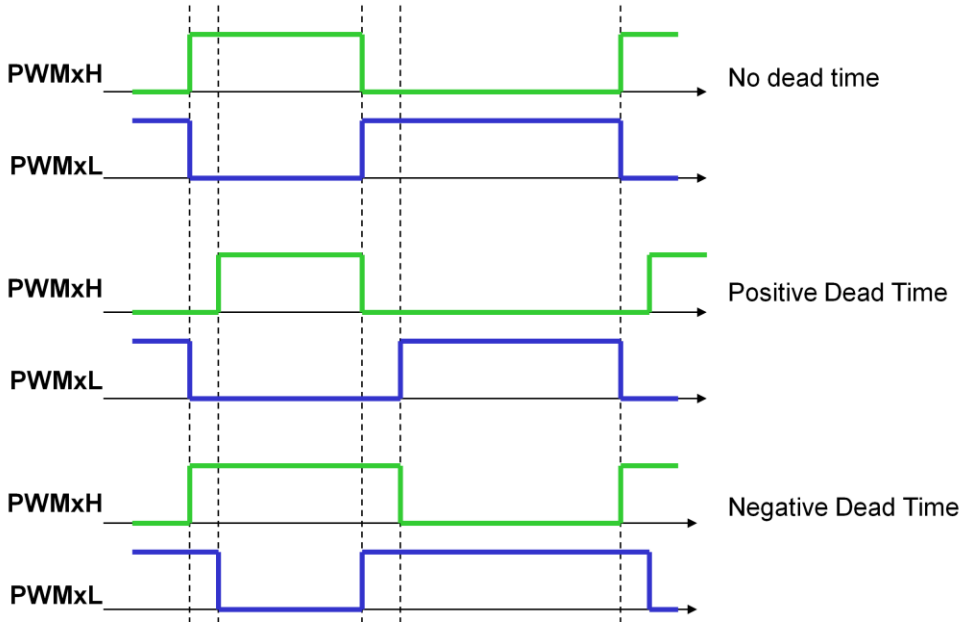
Dead time PWM

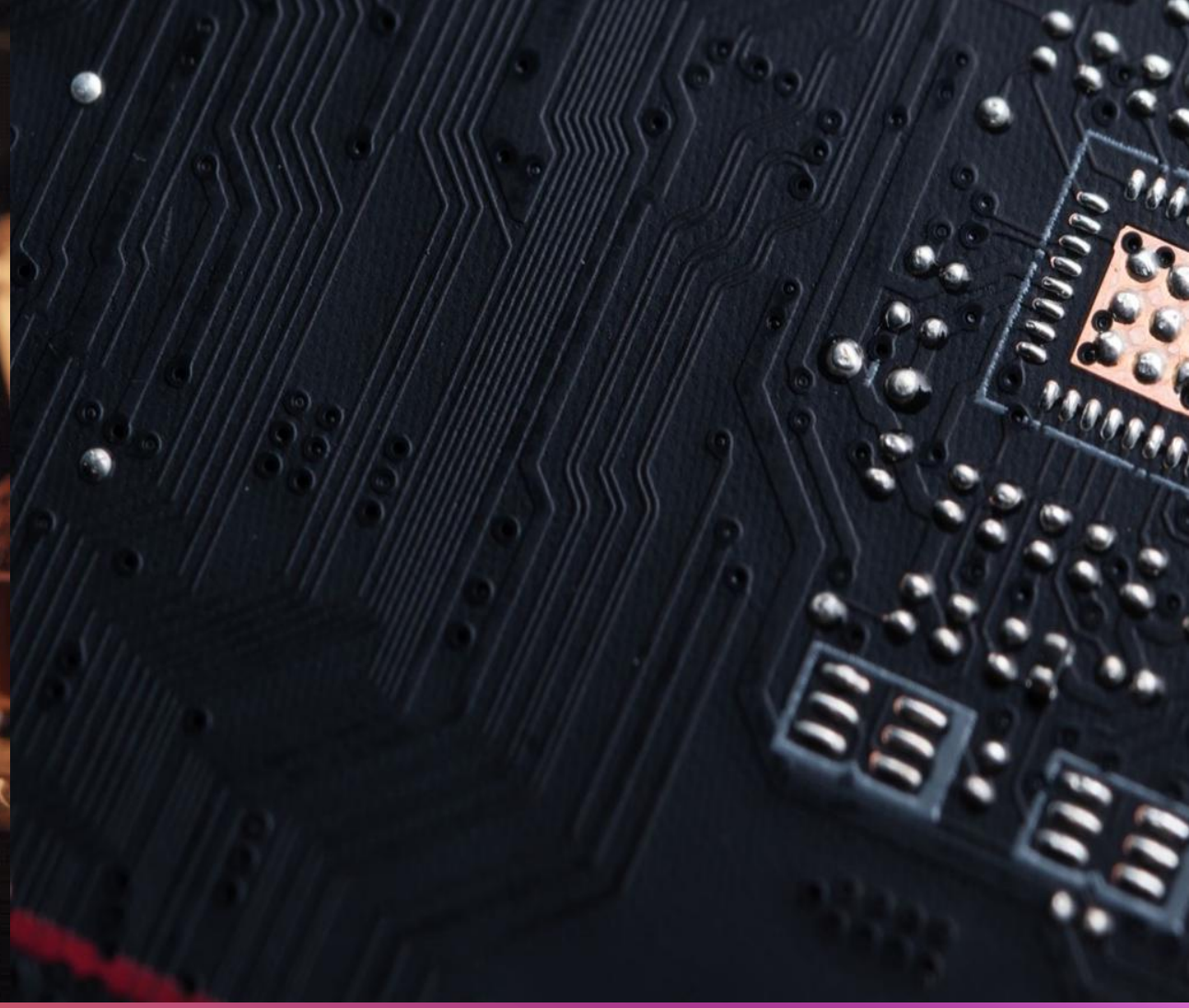
Приклад аудіопідсилювача class D



Джерело: Embedded (https://www.embedded.com/its-all-about-class/)

Dead time (мертвий час) у ШІМ використовується для забезпечення безпеки та ефективності роботи комутаційних пристроїв, наприклад транзисторів, в імпульсних перетворювачах та Н-мостах. Ось чому dead time важливий:
1.Запобігання короткому замиканню: Коли два транзистори працюють у парі, наприклад, у схемі Н-мосту, можливе одночасне включення обох транзисторів, що призведе до короткого замикання живлення через низький опір між ними. Dead time забезпечує, щоб один транзистор повністю вимкнувся, перш ніж інший увімкнеться.
2.Зменшення втрат потужності: Швидке перемикання транзисторів може спричинити великі струмові піки та втрати потужності. Введення dead time зменшує ці піки, дозволяючи транзисторам плавніше перемикатися.
3.Поліпшення надійності системи: Зменшення ймовірності одночасного включення комутаційних пристроїв збільшує надійність системи та знижує ризик пошкодження компонентів.
Нижче наведено приклад як виглядають сигнали ШІМ з мертвим часом:





Щиро дякую!