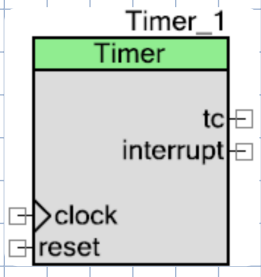
Timer

TopDesign



When ?

- Creating periodic interrupts
- Measuring → frequency / pulse of input signal
  → time between events
- Counting events

Principle : 8/16/24/32 bit register dec. at each CLK cycle

Ex : 16 bit register → 2^{16} values

24 MHz CLK freq.

$$TC\ freq. = \frac{clock\ frequency}{timer\ period} = \frac{24 \cdot 10^6\ Hz}{2^{16}} \simeq 366$$

So TC bit rise to 1 at f. of 366 Hz (We can use this event to control other events.)

⚠ How to choose the correct resolution btw. 8, 16, 24, 32 bits ?

$$TC\ period \leq 2^{resolution} \Leftrightarrow res = \left\lceil \frac{\log(timer\ period)}{\log(2)} \right\rceil$$ round ↑ to next 8 multipl.

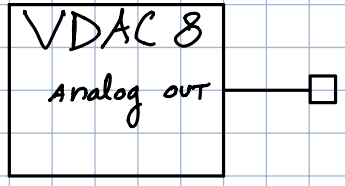
Code :

```
Timer_Start();
Timer_ISR_StartEx(Timer_ISR_Handler);
```

Timer_ReadCapture(); → inside handler.

DAC → Digital to Analog Converter (bits to voltage)

TopDesign :



Configuration :

Range : 0 - 1.020 V

Data Source : CPU

Speed : Slow

Strobe : Register Write

Code :

```
VDAC_Start();
for loop | VDAC_SetValue([0, 2^8 - 1]);
```

ADC → Analog to digital converter (voltage to bits)

Principle: Time quantization & Level quantization

(because cannot represent in finite # bits)

Performances depend on: Sampling freq. [Hz] & resolution [$\frac{V}{bit}$]

$\frac{\Delta \text{voltage}}{\#\text{bits coding}} \rightarrow 2^{(\text{reg. size})} - 1$

≡ Smallest detectable V change per bit.

Settings: Set input range (Vref) → Vdda 3.3V / Vdda 2 1.65V

Set input mode → single ended

Code:

```
ADC_Start();
ADC_StartConvert();
for(;;)
{
    if(ADC_IsEndConversion(mode)) {
        res = ADC_GetResult16(channel);
    }                                          // Polling mode
    voltage = ADC_CountsTo_Volts(channel, res);
    voltage = ADC_Read16();                    // Non polling mode
    purcentage = (voltage*100)/maxvoltage
}
```

Interrupts

1) Use "isr" component (interrupt service routine)
   a) Connect to the input pin
   b) Set falling edge

2) Build an interrupt handler

```
CY_ISR(ISR_Handler) {
    OPINNAME_Write(IPINNAME_Read());   // What we want to do when interrupted
    ISR_ClearInterrupt();
}
```

3) Start interrupt with custom handler:

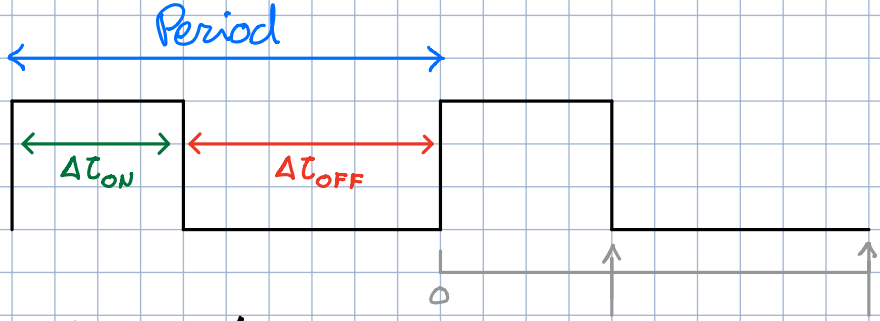
```
Cy GlobalInterruptEnable;
ISR_StartEx(ISR_Handler);
```

PWM → Pulse-Width Modulator

When?

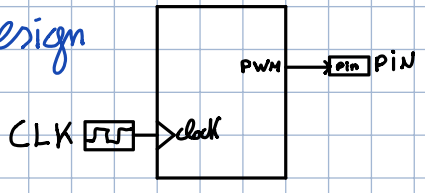
- Creating arbitrary square waves
- Changing brightness of a LED.
- Driving motors

How works?



$$\text{Duty cycle} = \frac{\Delta t_{ON}}{\text{Period}} = \frac{4}{10}\ \%$$

Top Design



Period calculations

$$\text{PWM period [s]} = \frac{\text{PWM period [/]}}{\text{CLK freq. [Hz]}}\ [s]$$

Ex₁: CLK f = 24MHz ⇔ $24 \cdot 10^6$ Hz

PWM period = $24 \cdot 10^3$

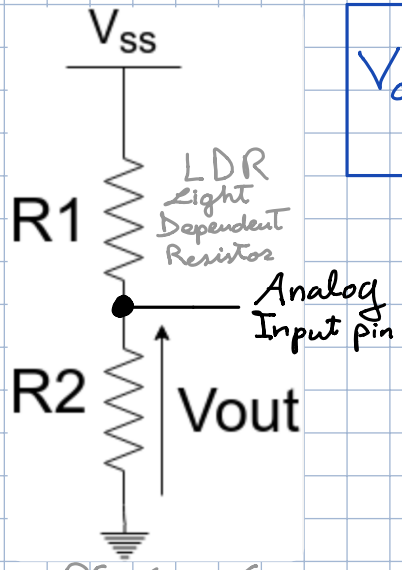
$$\frac{\text{CLK f.}}{\text{PWM p.}} = \frac{24\ 10^6}{24\ 10^3} = 10^3 \text{Hz} = 10^{-3} s = 1\ ms$$

Ex₁: period 100 μs = $100 \cdot 10^{-6} s$ = $10^{-4} s$

$$10^{-4} s = \frac{x}{10^6 \text{Hz}} \Leftrightarrow \text{period PWM} = 10^{-4} \cdot 10^6 = 100$$

Photoresistor

Analog Input Pin voltage ≡ Voltage divider V_{OUT}



$$V_{OUT} = \frac{R_2}{R_1 + R_2}$$

Protocol:

1) Measure LDR resistance under two conditions — Pitch black / Bright indoors

1) R_1 — Pitch black ~ 300 kΩ; Bright indoors ~ 3 kΩ

2) Choose R2 as to exploit ADC range as much as possible.
Hint: usually take R2 ~ R1 so that $V_{OUT} \simeq \frac{1}{2} V_{ss}$

2) $R2 = 10K\Omega$

3) Find min & max expected voltage at ADC

3) $min = \frac{1}{31} V_{ss} = 0,16V$ $\quad max = \frac{10}{13} V_{ss} = 3,84V$

LCD

TopDesign component name ≡ Character LCD

Pin connection: Requires 7 I/O pins

- Clear display
- Enable auto cursor inc.
- Reset cursor pos.

Code: (instance name)

```
LCD_Char_Start();
LCD_Char_Init();
```

Init for component normal work

for

```
LCD_Char_PrintString(char8 const string[]);
LCD_Char_ClearDisplay();
```

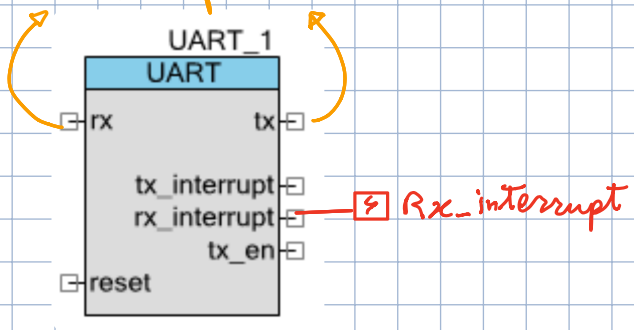
Eventally

```
LCD_Char_Position(uint8 row, uint8 column);
```

UART Why? To communicate with a PC

TopDesign:

GPIO ports



Configuration:

Baudrate : 9600

Data bits : 8

No parity bit

Stop bit : Just 1

Code:

```
CY_ISR(uart_rx_handler){

}

UART_Start();
rx_interrupt_StartEx(uart_rx_handler)
```

Send steing to the Tx buffer.

```
UART_PutString(const char8 string[]);
```