Interrupts and Timers

Chirag Sangani

Electronics Club
IIT Kanpur

Interrupts

- Low level programming concept.
- Extremely important – used extensively in modern computer programs.
- Irreplaceable.

Origin of Interrupts

- Normal view of a computer program: sequence of instructions executed serially, jumps are allowed.
- This view isn't good enough for the real world.
- Programs for embedded systems usually service real-life demands.
- Real-life demands don't wait for anything.

Origin of Interrupts

- Consider a typical embedded system program: it usually consists of an infinite loop, called the "program loop".
- In each iteration, the program checks whether events have occurred, gives suitable responses and performs periodic tasks.
- This model is sufficient if processor is extremely fast with respect to real world.

Example

```
-------
while(1){
        ----    ← Event 'A' handler
        ----
        ----
        ----
        ----    ← Event 'B' handler
        ----
        ----
        ----
        ----    ← Event 'A' occurs here
        ----
}
```

Why Interrupts?

- We need a method to handle events the moment they occur, and not after some delayed time.
- Interrupts are special events that can "interrupt" the normal flow of a program.
- The processor stops the normal program, handles the interrupt, and then resumes its normal work.

Example

```
main(){
        while(1){
                ----
                ----    ← Event ‘A’ occurs here
                ----
        }
}

handleA(){
        ----
        ----
}
```

Timers

- A timer is a register. Recall that registers are special, fixed-size variables with hardware implications.
- The timer, when started, begins at 0. After every time *t*, its value increases by 1.
- This process is **independent** of the CPU.
- When the timer reaches its maximum value, in the next cycle, its value becomes 0 again and the process repeats itself.

Timers

- Assume an 8 bit timer.

255 ← Maximum value

254

.

.

.

0 ← Starting value

Some statistics

- If the maximum value of a timer is *n* and clock period is *t,* then:
  1. Timer cycle period $= (n+1) \times t$
  2. Frequency of timer $= f = \frac{1}{t}$
  3. Frequency of timer cycle $= \frac{1}{(n+1)\times t}$

Timers and Interrupts

- Timers can generate certain interrupts: two, to be precise.
- These are called OVERFLOW interrupt and COMPARE MATCH interrupt.

OVERFLOW interrupt

- OVERFLOW is generated when a timer tries to exceed its maximum value and resets to 0.
- The name is derived from the fact that the timer has “overflowed” its limit.
- The interrupt may or may not have a handler. In either case, the timer continues to run; remember: timers are **independent** of the CPU.

OVERFLOW statistics

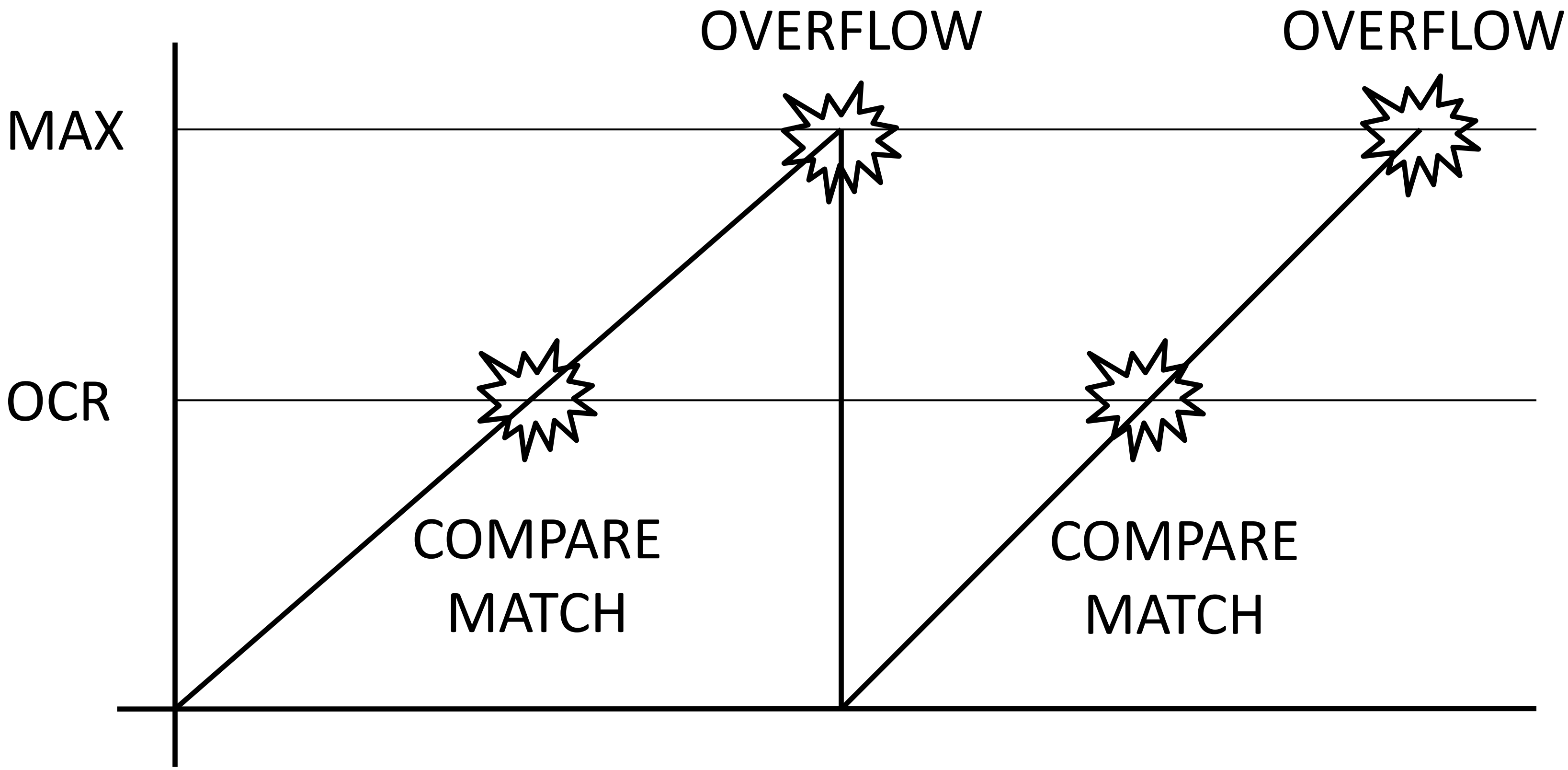
- Suppose a timer of maximum value *n* has a time period *t* (also called as clock period).
- Then the timer cycle frequency $= \frac{1}{(n+1) \times t}$
- If OVERFLOW interrupt is enabled, then an interrupt is generated in every cycle.
- Thus, OVERFLOW interrupt frequency $= \frac{1}{(n+1) \times t}$

Electronics Club IIT Kanpur

COMPARE MATCH interrupt

- There is a register called as **OCR** (Output Compare Register), whose value we can set.
- After every clock period, the timer is incremented by 1 (or reset to 0 in case it is at maximum value).
- Before incrementing, the value of the timer is compared to **OCR**. If the two are equal, a COMPARE MATCH interrupt is generated.

OVERFLOW and COMPARE MATCH



COMPARE MATCH statistics

- Suppose a timer of maximum value *n* has a time period *t* (also called as clock period).
- Then the timer cycle frequency $= \frac{1}{(n+1)\times t}$
- If COMPARE MATCH interrupt is enabled, then an interrupt is generated in every cycle.
- Thus, COMPARE MATCH interrupt frequency $= \frac{1}{(n+1)\times t}$

Summary of Timers

- A timer is not affected by interrupts: it generated interrupts, but it does not stop running because of them.
- Interrupts is how timers are useful. Sample applications: digital clock, periodic events (such as blinking LEDs quickly for POV globe), etc.

Timer Modes

- A timer works in three modes: Normal, CTC and PWM.
- All three modes are again unaffected by interrupts, but all three modes can generate interrupts.
- The timer mode used so far in this presentation is normal mode.

Normal Mode

- Standard mode: Timer starts at 0, goes to maximum value and then resets itself.
- OVERFLOW and COMPARE MATCH interrupts generated as normal.

CTC (Clear Timer on Compare) Mode

- Timer starts at 0 as usual, but instead of resetting after maximum value, it resets after reaching value specified in **OCR** register.

OCR	← Maximum Value
OCR – 1	
.	
.	
0	← Starting Value

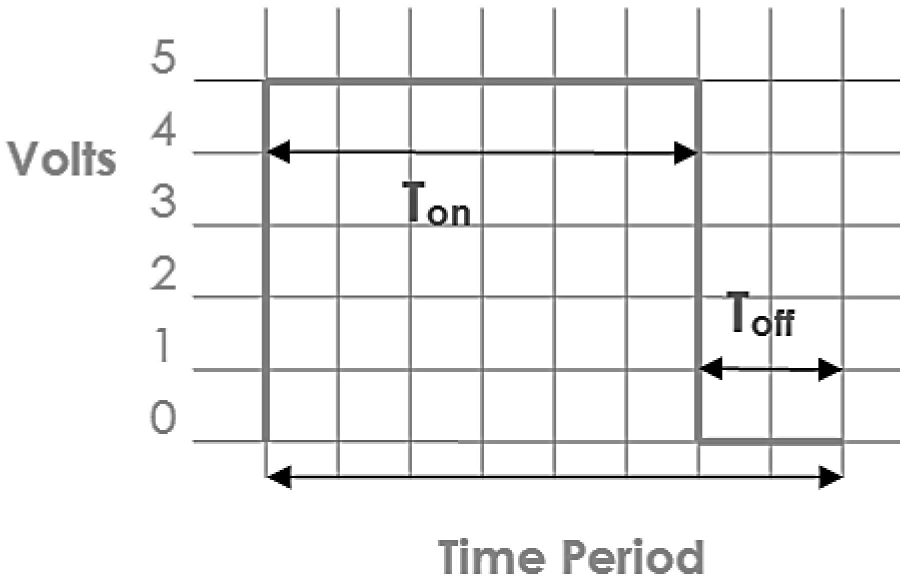
CTC mode statistics

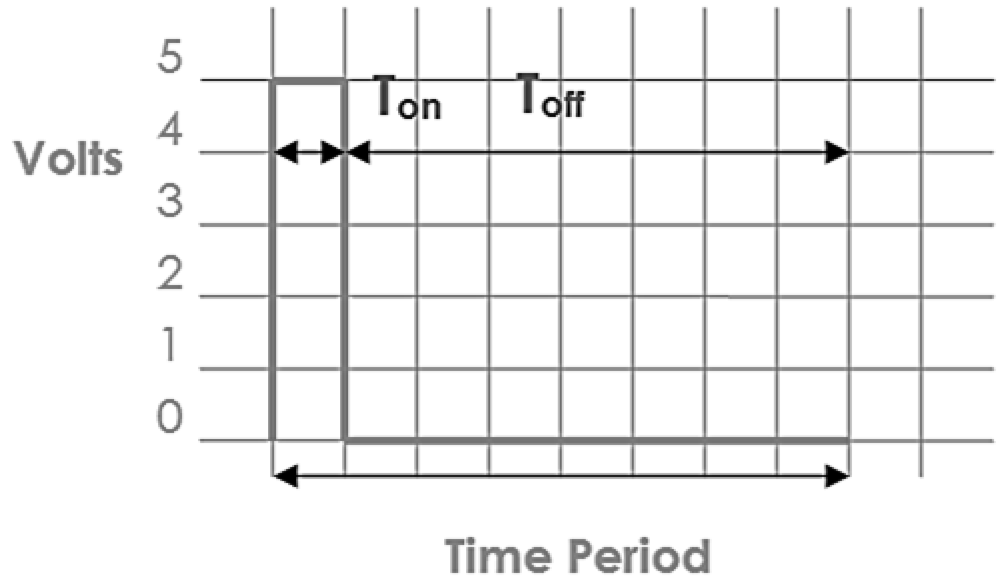
- If clock time period is t:
  1. Timer cycle time period $= (OCR + 1) \times t$
  2. Frequency $= \frac{1}{(OCR+1)\times t}$
- COMPARE MATCH interrupt will work normally, but OVERFLOW interrupt will not work (Why?).

PWM (Phase Width Modulation) Mode

- Simple method of obtaining analog output of any value between 0 and 5V.
- Suppose desired output is *x*% of 5V. If, for a time period *t*, the output is 5V for *x*% time and is 0 for the remaining time, then average voltage is x% of 5V.
- If this time period is extremely small and the process is repeated continuously, then output behaves as analog value.

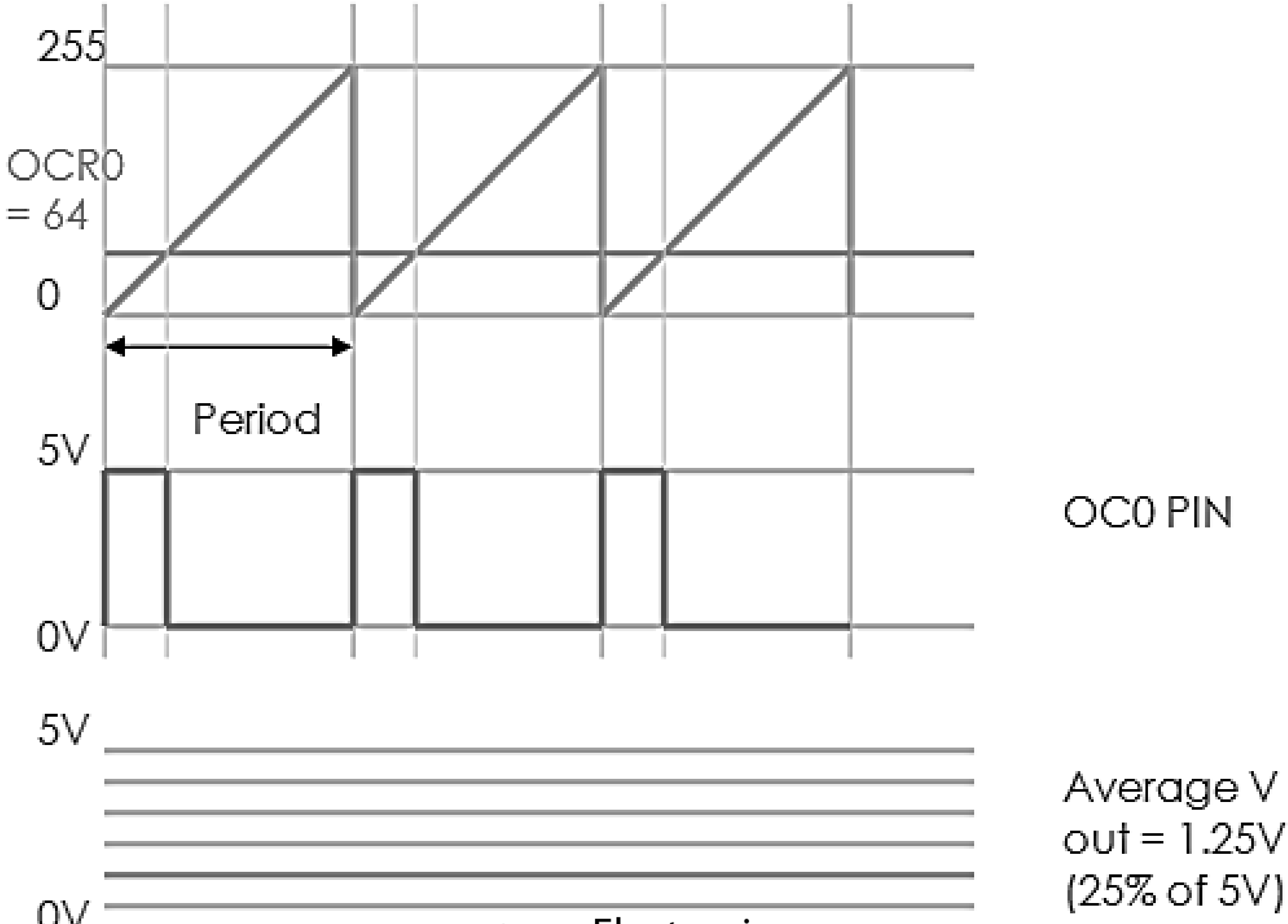
PWM mode


Volts
5
4
3
2
1
0
T_{on}
T_{off}
Time Period


Volts
5
4
3
2
1
0
T_{on}
T_{off}
Time Period

PWM mode

- This “analog” value is obtained using timers.
- A specific pin is set as output. When the timer reaches 0, the voltage of the pin is set to 5V.
- When the timer reaches the value specified by OCR, on the next clock, the pin voltage is set to 0 until the timer resets itself.



Electronics Club
IIT Kanpur

PWM statistics

- If clock time period is t and maximum timer value is n:
  1. Timer cycle time period $= (n + 1) \times t$
  2. Frequency $= \frac{1}{(n+1)\times t}$
  3. Duty cycle $= \frac{OCR+1}{n+1} \times 100\%$
  4. Output voltage $= \frac{OCR+1}{n+1} \times 5V$
- COMPARE MATCH interrupt and OVERFLOW interrupt will work properly.

Thank you