Clocks and Timers

The Embedded Discovey Book

What is Clock ?

- **Computers use an internal clock to synchronize all of their calculations.**
- **The clock ensures that the various circuits inside a computer work together at the same time.**

What is Clock ?

- **Clock speed is measured by how many ticks per second the clock makes.**
- **The unit of measurement called a hertz (Hz).**

For loop isn't good for creating delays

```
#[inline(never)]
fn delay(tim6: &tim6::RegisterBlock, ms: u16) {
    for _ in 0..1_000 {}
}
```

No Operation

There is a way to prevent LLVM from optimizing the for loop delay: add a *volatile* assembly instruction. Any instruction will do but NOP (No OPeration) is a particular good choice in this case because it has no side effect.

Your for loop delay would become:

```
#[inline(never)]
fn delay(_tim6: &tim6::RegisterBlock, ms: u16) {
    const K: u16 = 3;
    for _ in 0..(K * ms) {
        aux9::nop()
    }
}
```

Hardware Timer

The basic function of a (hardware) timer is ... to keep precise track of time. A timer is yet another peripheral that's available to the microcontroller; thus it can be controlled using registers.

TIM6

We'll be using one of the *basic* timers: TIM6. This is one of the simplest timers available in our microcontroller

Registers of TIM6 Peripheral

The registers we'll be using in this section are:

- **SR,** the status register.

Registers of TIM6 Peripheral

The registers we'll be using in this section are:

- **SR,** the status register.
- **EGR,** the event generation register.

Registers of TIM6 Peripheral

The registers we'll be using in this section are:

- **SR,** the status register.
- **EGR,** the event generation register.
- **CNT,** the counter register.

Registers of TIM6 Peripheral

The registers we'll be using in this section are:

- **SR,** the status register.
- **EGR,** the event generation register.
- **CNT,** the counter register.
- **PSC,** the prescaler register.

One Pulse mode

Step 1:

Set the timer trough auto reload register **ARR**

One Pulse mode

Step 2:

We have to enable the counter register

(CR1.CEN = 1)

One Pulse mode

Step 3:

We have to reset the value of count register **CNT** to zero.

On each tick it’s value get incremented.

One Pulse mode

Step 4:

Once the CNT register has reached the value of the ARR register, the counter will be disabled by hardware

(CR1.CEN = 0)

and an *update event* will be raised

(SR.UIF = 1).

CNT register increase at a Frequency

CNT register increase at a frequency of **apb1 / (PSC +1)** times per second.

Also note : **1 Khz = 1 ms (period)**

Configure the prescaler to have the counter operate at 1 KHz

Where:

Formula: **apb1 / (PSC +1)**

APB1_CLOCK = 8 MHz

PSC = ?

8_000_000/ (PSC + 1) = ?

Configure the prescaler to have the counter operate at 1 KHz

Where:

Formula: **apb1 / (PSC +1)**

APB1_CLOCK = 8 MHz

PSC = ?

8_000_000/ (7_999 + 1) = 1_000 Hz *OR* 1kHz

The counter (CNT) will increase on every millisecond