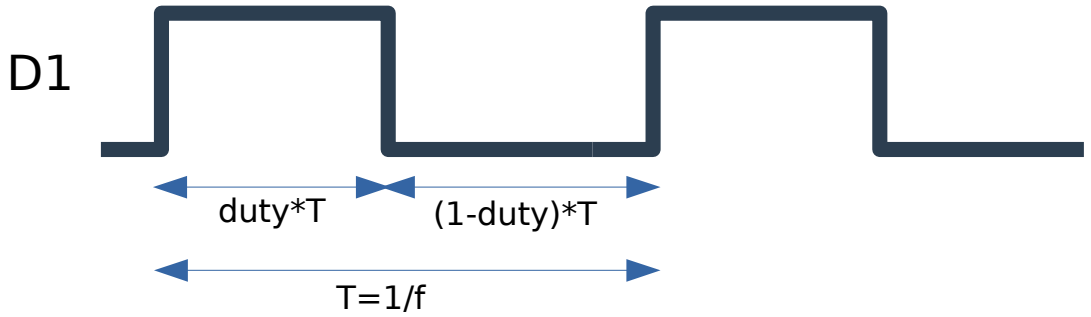
# Vaja 4

Delo s časovnikom (timer)

# Naloga

**Napišite program, ki bo na eni od diod LED generiral  pravokotni vlak pulzov z dano frekvenco in obratovalnim ciklom (duty cycle)**




Na prikazovalniku LCD naj bosta ves čas prikazana trenutna frekvenca in “duty cycle”

S tipkama T1, T2 naj bo frekvenca nastavljiva v območju 1-20 Hz (korak 1Hz)

S tipkama T3, T4 naj bo duty cycle nastavljiv v območju 5-95% (korak 5%)

# Različni pristopi

## 1. z zakasnitvijo (npr. z ASF modulom delay)

Težava je nenehno čakanje na potek določenega časa, ki ustavi vse druge operacije

## 2. s povpraševanjem (polling), izvedba z zanko

V zanki opazujemo stanje časovnika in se odzovemo, ko je potrebno (vmesni čas se lahko izvajajo druga opravila oz. glavni program)

Natančnost odziva je odvisna od trajanja opravil v zanki

Uporabimo prosto tekoči števec (časovnik, timer), ki bo meril čas

## 3. s prekinitvami časovnika (timer)

Časovnik sam javi, kdaj je potekel določen čas (med tem teče glavni program)

Ob danem trenutku se glavni program prekine → izvede se prekinitvena rutina (ISR – interrupt service routine), nato se nadaljuje glavni program

# Časovnik - timer

**Modul sam/drivers/tc**

Dodajte modul v projekt (glej vajo 1)

**3 časovniki (TC0, TC1, TC2 → glej sam3x8e.h)**

32 bitni števci

Vsak vsebuje 3 neodvisne kanale => skupaj 9 števcev

Uporabili bomo TC0, kanal 0

**Za periferne enote je potrebno najprej vklopiti uro**

**static inline void sysclk_enable_peripheral_clock(uint32_t ul_id)**

Glej modul "common/services/clock"

ul_id: ID_TC0 (glej sam3x8e.h)

# ASF modul sam/drivers/tc

## Funkcije:

### void tc_init(Tc *p_tc, uint32_t ul_Channel, uint32_t ul_Mode)

p_tc: TC0 (sam3x8e.h)

ul_Channel: 0

ul_Mode: bitna maska z nastavitvami (glej component_tc.h, datasheet, register TC0_CMR - channel mode reg.)

- Delilnik ure nastavimo na 32 (maska TC_CMR_TCCLKS_TIMER_CLOCK3)
- izberemo "waveform" način (maska TC_CMR_WAVE)

### void tc_enable_interrupt(Tc *p_tc, uint32_t ul_channel,uint32_t ul_sources)

Potrebujemo za drugi del vaje → izvedba s prekinitvami

ul_sources: maska virov prekinitev, glej TC_IER register

Potrebujemo le prekinitev, ko števec doseže vrednost v reg. RC → maska TC_IER_CPCS

# ASF modul sam/drivers/tc

## Funkcije:

**void tc_write_rc(Tc *p_tc, uint32_t ul_channel,uint32_t ul_value)**

Vpis vrednosti za reset števca in proženje prekinitve v primerjalni register RC
Potrebujemo samo pri izvedbi s prekinitvami

**void tc_start(Tc *p_tc, uint32_t ul_channel)**

Vklop števca

**void tc_stop(Tc *p_tc, uint32_t ul_channel)**

Ustavitev števca

**uint32_t tc_read_cv(Tc *p_tc, uint32_t ul_channel)**

Branje vrednosti števca

# Vaja4b: izvedba s prekinitvami

**Prekinitev (interrupt) je poseben signal, ki prekine izvajanje glavnega programa in izvede prekinitveno rutino (ISR, interrupt service routine)**

- Prekinitev lahko sproži vsaka periferna enota
- Ob prekinitvi se trenutno stanje procesorja shrani, po koncu ISR se stanje obnovi, da se lahko nadaljuje prekinjen program
- Prekinitve lahko imajo prioritete – v primeru sočasnih zahtev po prekinitvi s strani različnih perifernih enot
- Za prekinitve skrbi nadzornik prekinitev (pri SAM3X8E je to NVIC - nested vectored interrupt controller)
- Prekinitve je potrebno omogočiti na dveh nivojih: NVIC in preiferna enota

**Pri inicializaciji TC0 je potrebno vklopiti prekinitve in reset števca ob doseženi vrednost (podana v primerjalnem registru RC →  število ciklov števca za predpisan časovni interval)**

- Dodati je potrebno masko za TC_CMR register TC_CMR_WAVSEL_UP_RC → glej tc_init(…)

**Konfiguracija NVIC (glej thirdparty/CMSIS/include/core_cm3.h)**

- **__STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)**
- NVIC - nested vectored interrupt controller = vektorski nadzornik gnezdenih prekinitev)
- IRQn:  oznaka prekinitve (glej sam3x8e.h)

# Prekinitve

## Prekinitvena rutina (ISR – interrupt service routine)

Deklaracije (prototipi) so že definirane za vse prekinitve (glej “sam3x8e.h”)

Definicijo rutine bomo zapisali sami v “main.c”

```
void TC0_Handler(){
...
}
```

Pred koncem ISR je potrebno pobrisati zastavice prekinitve

- Da lahko enota sproži naslednjo
- Različne periferne enote imajo različen način
- Za timer je potrebno prebrati statusni register števca

**uint32_t tc_get_status(Tc *p_tc, uint32_t ul_channel)**