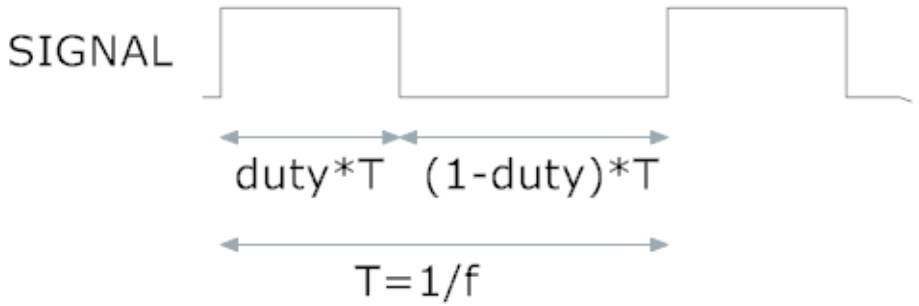
Excercise 3
Timer, polling, interrupts

Write a program that toggles one of the LED pins (e.g. LED0 on pin P0.4) to produce a square signal.



Additional features: frequency (f) and duty cycle (duty) should be adjustable with keys and displayed on the LCD.

T0, T1 : used to adjust the frequency from 1-10 Hz (with 1Hz step)
T2, T3: used to adjust the duty cycle from 1-99 % (with 1% step)

TIMER1(lpc2138 also has TIMER0)

Registers (see timer.h):
T1TC – timer counter – counts divided peripheral bus (VPB) clock transitions
T1PC – prescale counter – counts VPB clock transitions
T1TCR – timer control – writing *counter_enable (counter_reset)* **starts/stops the counter**
T1MRj – match registers – when T1TC reaches a value of any T1MRj (j=0,1,2,3), a predefined action can be performed (trigger interrupt, reset the counter, stop the counter)

void timer1_init(int prescale, int *match, int control, int count, int pin);

prescale : when T1PC == prescale => T1TC = T1TC+1
match : an array with 4 integers for match registers: when T1TC == match[j] => perform an action
e.g. Int match[4]={100,200,0,0}
control : configure the actions for match events (T1TC == T1MRj for j=0,1,2,3)
mrji : T1TC == T1MRj =>trigger interrupt
mrjr : T1TC == T1MRj => reset the counter
mrjs : T1TC == T1MRj => stop the counter
e.g. control = mr0i|mr0r => when T1TC==T1MR0 => trigger interrupt and reset the counter
count : count mode
timer : count VPB clock transitions
counter_rising | capi : count positive transition of preconfigured capture pin
counter_falling | capi : count negative transition of preconfigured capture pin
counter_both | capi : count all transition of preconfigured capture pin
capi = cap0, cap1, cap2, cap3 (bitmasks for 4 available capture signals)
pin : which pin to use as capture signal

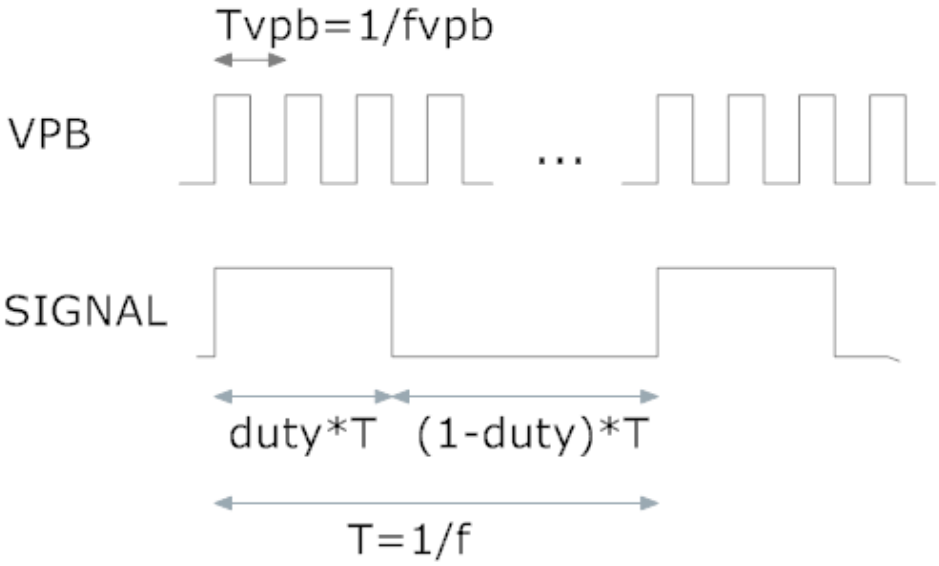
First approach (polling):

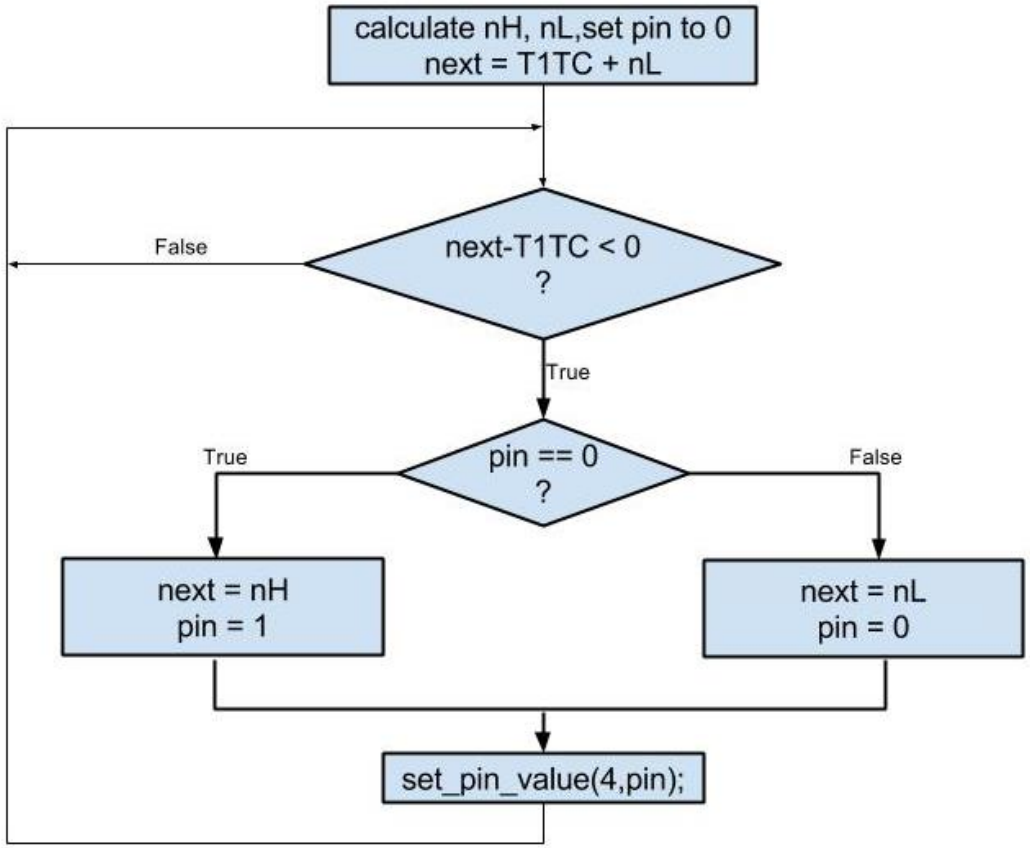
Initialize the timer (startup.c) :

```
int prescale = 0;
int match[] = { 0, 0, 0, 0 };
int control = 0;       /*  we dont want any special action when T1TC = match[j]  */
int count = timer;     /* timer1 will be free flowing counter of vpb clock */
int pin = 0;           /* no capture pin */

timer1_init( prescale, match, control, count ,pin);

/* To start the timer: */
T1TCR = counter_enable;
```

First approach (polling):

Main program (main.c):





nH = duty * fvpb / f
nL = (1 - duty) * fvpb / f

Second approach (interrupts):

We need to configure the timer to trigger interrupts and the controler that will handle them.

void timer1_init(int prescale, int *match, int control, int count);

prescale = 0;	**we are still counting VPB clock**
match = {xxx, 0, 0, 0};	**we need to set the alarm for our timer**
control = mr0i | mr0r;	**when T1TC==match[0] => trigger interrupt and reset the counter**
count = timer;	**timer is still normal free flowing counter of VPB clock**

Vectored interrupt controler (VIC):
Possible interrupt sources (see vic.h for available bit masks to insert into the interrupt control register) :

wdt, arm_core0, arm_core1, timer0, timer1, uart0, uart1, pwm0, i2c0, spi0, spi1_ssp, pll, rtc, eint0, eint1, eint2, eint3, ad0, i2c1, bod in ad1

void vic_init(int fiq, int irq, voidfuncptr *functions, int *interrupts, voidfuncptr def);

fiq	**: fast interrupts –bitmask combination (using bitwise OR operator)**
irq	**: regular priority based interrupts–bitmask combination (using bitwise OR operator)**
functions	**: array of 16-ih function pointers for priority based vectored IRQ interrupts**
interrupts	**: array of 16-ih priority based interrupts that will trigger the corresponding function**
def	**: default function pointer for the non-vectored interrupt**

Second approach (interrupts):

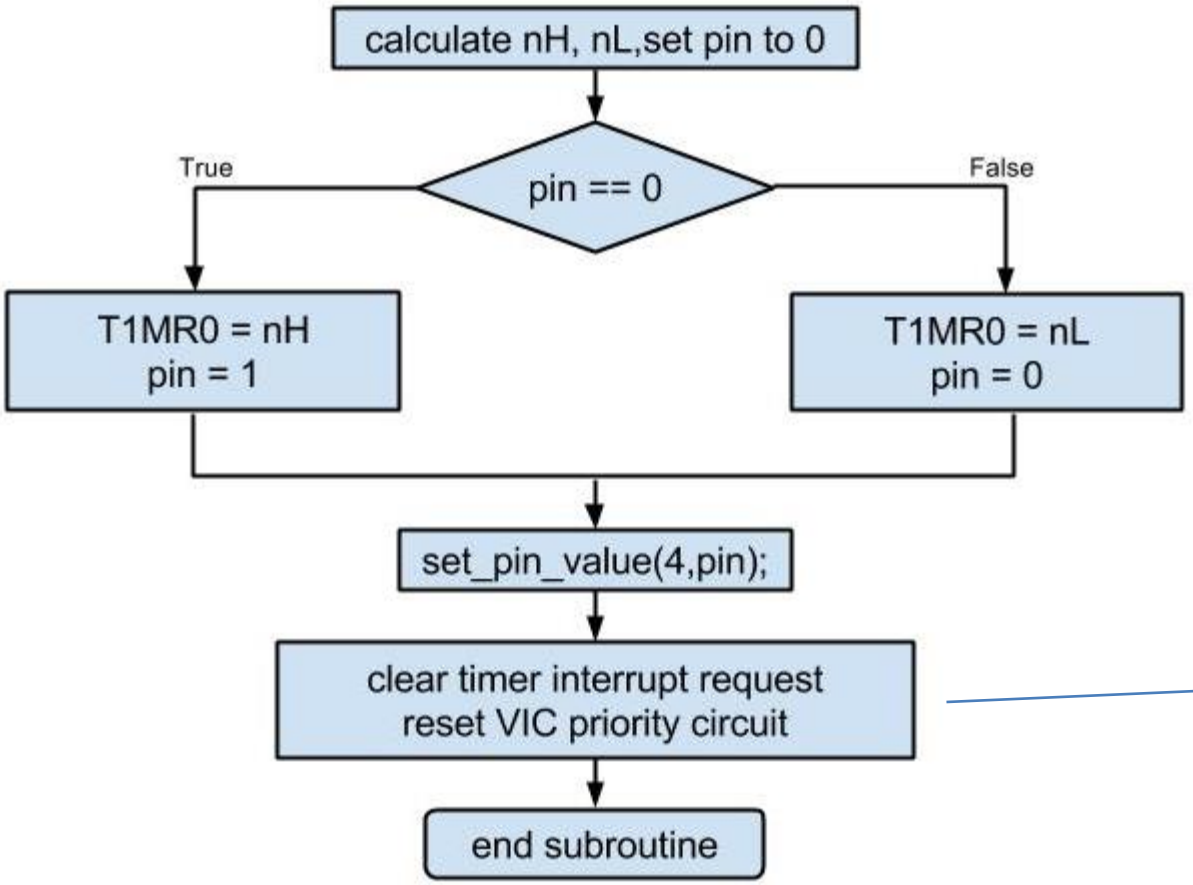
VIC initialization (startup.c) :

```
int fiq = 0;              //we wont need fast priority interrupt
int irq = timer1;         //we will enable timer1 interrupt handling

/* array of 16 function pointers. Only the first slot will be filled with the function
address to call when timer1 triggers the interrupt*/
voidfuncptr function[16] = {pulse_train, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

/* array of 16 vectored interrupts, that are arranged according to their priority. First
slot has the highest priority. We only need one interrupt here (timer1) */
int interrupt[16] =
{ timer1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
```

=> vic_init(…)

Second approach (interrupts):

pulse_train(): **IRS (interrupt service routine) (we will define it in main.c):**



T1IR=mr0_interrupt;
VICVectAddr=0;

Usefull functions:

Initialize the microcontroler (see init.h, init.c):

void init(int clock_mhz, int div,int input,int output,int value);

- **clock_mhz: cpu clock frequency fclk in mHz (12,24,36,48, or 60)**
- **div: peripheral bus frequency divider**
- **cclk: fvpb = fclk**
- **cclk_2: fvpb = fclk/2**
- **cclk_4: fvpb = fclk/4**
- **input, output: bit masks for input and output pins**
  - **use P0_xx macros (defined in gpio.h) to construct bit masks, e.g.:**
  - **input = P0_12 | P0_13**
  - **output = P0_4 | P0_5**
- **value: initial output pins value**

You only need to call this function once.

Usefull functions:

GPIO functions (see gpio.h, gpio.c):

void set_pin_value(int pin_num, int pin_value);
Write digital value to the output pin with pin number pin_num (0,1,...15).

int get_pin_value(int pin_num);
Read digital value of the input pin with pin number pin_num (0,1,...15).

void wait(int delay_us);
a delay function (waits for delay_us microseconds)

Usefull functions:

Writing to the LCD (see main.c):

void lcd_driver();

Call this function to refresh the display. Use lcd_string pointer to write text to LCD buffer.

For example to write text to LCD:

```
strcpy(lcd_string, "Hello"); // strcpy() is part of standard string.h library
lcd_driver();
```