Operation of an LED

In order to be able to turn on/off an LED with a specific value in the period, e.g. 10ms, we must first set the "Direction" of the specific Pin as "Output". Initially, the PORT we will use is PORTD as its first four PINs are connected to LEDs, on our board (as shown in the Board Overview). To set a PIN as "Output" we need to set the 5th bit of the DIR (Data Direction) register to '1'. Now we can assign a value of '0' or '1' to the Out (Output Value) Register and "turn on" or "turn off", the LED respectively.

Note: In the Simulation phase the times of the "delay" as well as the "timers" (which we will see later) are not representative of the time we expect. Only by running the code on the board can we understand the actual times. For this reason, we can choose relatively short times, to avoid long waiting delays.

The code of the example implementation is shown below:

```
#include <avr/io.h>
#include <util/delay.h>

#define del 10

int main(void){
	PORTD.DIR |= PIN1_bm; //PIN is output
	PORTD.OUT |= PIN1_bm; //LED is off
	while (1) {
		PORTD.OUTCLR= PIN1_bm; //on
		_delay_ms(del); //wait for 10ms
		PORTD.OUT |= PIN1_bm; //off
		_delay_ms(del); //wait for 10ms
	}
}
```

You can use the example code to run the "Simulation" procedure in "Microchip Studio". Open the I/O window (Debug => Windows => I/O) and start debugging. Execute the commands step by step (the delay function is not executed step by step) and observe the values taken by the registers of PORTD.

Note: To see the values of the declared PIN1_bm and other constants that we will see next, you can refer to the header file iom4808.h, which is located in the folder where you have Microchip Studio installed:

e.g. Path, “~\Studio\7.0\packs\atmel\ATmega_DFP\1.6.364\include\avr\iom4808.h”

Interrupt

The switches that can be used are located in the PORTF and are PIN5 and PIN6 (see Board Overview). To use a switch, the Pullup enable bit corresponding to it (refer to page 158 in the ATmega4808 DataSheet). Also, we need to select at which point in the pulse the management module of the interrupt. Here we choose to enable it at both ends of the pulse (refer to p. 158 in the ATmega4808 DataSheet). By enabling the specific bits of PIN5 the system can accept an interrupt when the switch is pressed.

When the interrupt occurs we need to be directed to a function that will handle it. This function is an ISR Routine which takes as an argument the one interrupt vector which corresponds to an interrupt. For example the interrupt vector for the PINF interrupt is PORTF_PORT_vect. In the iom4808.h file there are all the interrupt vectors that can be be used by our board.

The code of the example implementation, is shown below:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define del 10
int interr=0; //logic flag

int main() {
    PORTD.DIR |= PIN1_bm; //PIN is output
    PORTD.OUT |= PIN1_bm; //LED is off
//pullup enable and Interrupt enabled with sense on both edges
    PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;
    sei(); //enable interrupts
    while (interr==0) {
        PORTD.OUTCLR= PIN1_bm; //on
        _delay_ms(del);
        PORTD.OUT |= PIN1_bm; //off
        _delay_ms(del);
    }
cli(); //disenable interrupts
}
```

```
ISR(PORTF_PORT_vect){

//clear the interrupt flag

int intflags = PORTF.INTFLAGS;

PORTF.INTFLAGS=intflags;

interr=1;

}
```

In our system we have an LED and a Switch. Normal operation sets the LED to flash with a frequency of 10ms. When the switch is pressed the interrupt is activated and I change the value of the interr variable to stop the LED from flashing and stop the operation.

To enable the interrupt you must press the 5th bit of the INTFLAGS register in PORTF, if the program is at a breakpoint of your choice. Once you change the bit from '0' to '1' and press the program to go to the next instruction (STEP OVER), the interrupt routine will be activated and you will see that we are now in this function.

Timer

In order for Timer TCA0 to operate in normal mode and to execute an interrupt when it reaches a predicted value it must:

- Assign the value '0' to the CTRLB Register (Normal Mode).

- Set the value '0' in the CNT Register (set the timer to zero).

- Give the predicted value to the CMP0 Register.

- Set the Clock Frequency and activate it via the CTRLARegister.

- Finally, enable the Interrupts through the INTCTRL Register.

Note: For more details you can refer to p. 243, in ATmega4808. DataSheet.

The counter will start counting and when it reaches the value set in CMP0, the ISR Routine will be activated with the Vector TCA0_CMP0_vect argument. This is the Interrupt Vector set for this Timer.

The code of the example implementation, is shown below:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define ped 20

int interr=0;

int main() {
  PORTD.DIR |= PIN1_bm; //PIN is output
  PORTD.OUTCLR= PIN1_bm; //LED is on

//(σελ 219, 224, 205) 16-bit counter high and low
  TCA0.SINGLE.CNT = 0; //clear counter
  TCA0.SINGLE.CTRLB = 0; //Normal Mode (TCA_SINGLE_WGMODE_NORMAL_gc  σελ 207)
  TCA0.SINGLE.CMP0 = ped; //When reaches this value -> interrupt CLOCK FREQUENCY/1024
  TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc; //(= 0x7<<1 σελ 224 )
  TCA0.SINGLE.CTRLA |=1;//Enable
  TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm; //Interrupt Enable (=0x10)
  sei(); //begin accepting interrupt signals
  while (interr==0) {
  }
  PORTD.OUT |= PIN1_bm; //LED is off
  cli();
}
```

```
ISR(TCA0_CMP0_vect){

    TCA0.SINGLE.CTRLA = 0; //Disable
    //clear flag
    int intflags = TCA0.SINGLE.INTFLAGS;
    TCA0.SINGLE.INTFLAGS=intflags;
    interr=1;
}
```

We turn on the LED and activate the timer. When the timer reaches the value we have initially set, then:

- The interrupt management routine is activated.

- We change the interr variable.

- Our program is completed by turning off the LED.

Note: You can use breakpoints to see if an interrupt is triggered.

The value of the CMP0 register (value) defines the time interval until the timer interrupts (generally ends and starts over).

The formula for the calculation is below:

$$f_{timer} = \frac{f_{system}}{\mathrm{N_{prescaler}}}$$

$$value = T * f_{timer}$$

The ATmega4808 operates at a maximum of 20 MHz and $Nprescaleer$ takes the value we have set for clock frequency. For example in our code we have set a value of 20 so the time of timer is:

$$f_{timer} = \frac{20MHz}{1024} = 19{,}531\ KHz$$

$$T = \frac{value}{f_{timer}} = 1{,}024\ ms$$

Laboratory Exercise 01:

Traffic Lights

The purpose of the Lab Exercise is to simulate the operation of a motorway interchange, which consists of a major road and a minor road. On the major road there is a traffic light for cars and a traffic light for pedestrians, which is only lit after a push button is pressed.

On the small road there is a sensor, (which will be implemented with a Random function), which when it detects that there is a waiting car, sends a command to turn on the red light for the big road and the green light for the small road. The smaller road has no smart pedestrian light.

When the pedestrian button is pressed for the smart traffic light on the big road, first the red traffic light for cars on the big road turns on, then the green traffic light for cars on the small road turns on (if it is already on, leave it on) and finally the green traffic light for pedestrians on the big road turns on for a certain period of time.

For the implementation of Lab Exercise 1, consider that:

- The traffic light is green when the LED is on and red when it is off. The three PORTD PINs used are PIN0, PIN1and PIN2.

- Simulate the sensor with the random function, i.e. when random gives a random number ending in 0, 5 or 8 assume that there is a car on the small road. No need to make an interrupt, a simple if is enough.

- With an interrupt, the pedestrian button press must be implemented. The use of PIN5 of PORTF will be used. With timer and interrupt the time at which the pedestrian light is on can be implemented. You define a unit of time, which is convenient, for the simulation, but explain the thinking and the process of calculating the value.