13th-Dec-2020

Winter term 2020/21

Networked Embedded Systems

Homework 02 : Energy Consumption

<u>Task 1:</u> <u>Toggle red LED for every 1s</u>

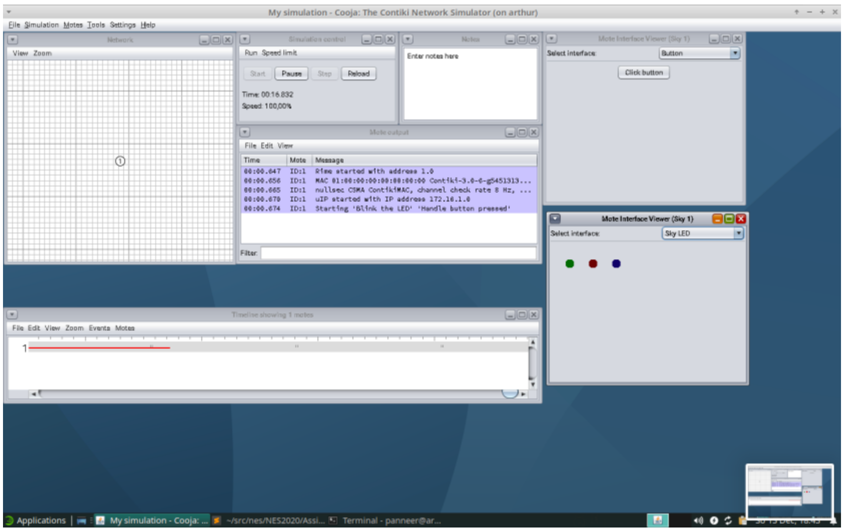
We implemented the protothread (led_pt) which starts automatically.

- A Timer from etimer module is initiated for every second using the CLOCK_SECOND constant and the function etimer_set().
- process waits for the etimer to get expired. This was verified using the function etimer_expired().
- Each time the etimer is expired the red LED is toggled using the function leds_toggle(LEDS_RED)
- After toggling, the timer is again reset to CLOCK_SECOND using the function etimer_set().

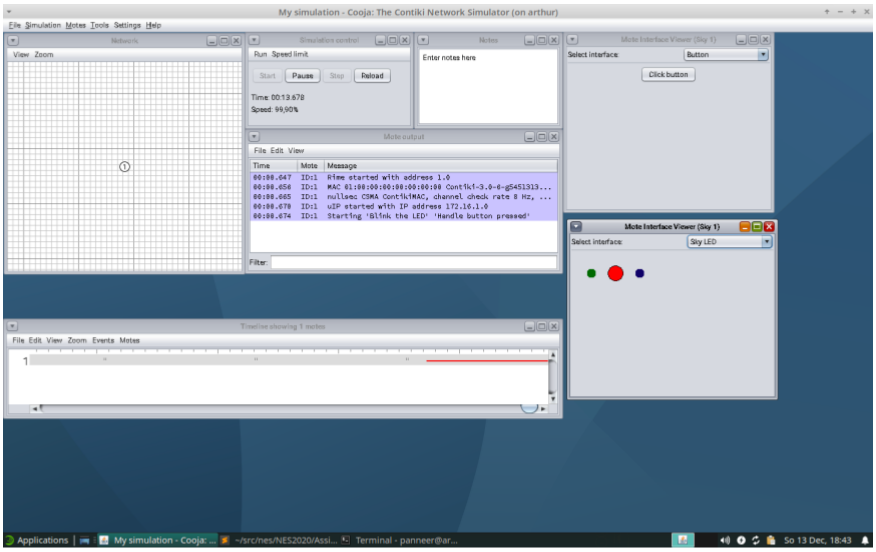
Protothread Implementaion:

```
PROCESS_THREAD(led_pt, ev, data) {
    static struct etimer timer;
    PROCESS_BEGIN();

    etimer_set(&timer, CLOCK_SECOND);


   while(1) {
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
       leds_toggle(LEDS_RED);
       etimer_set(&timer, CLOCK_SECOND);
   }

    PROCESS_END();

}
```

Output 1: led off



Output 2: led on



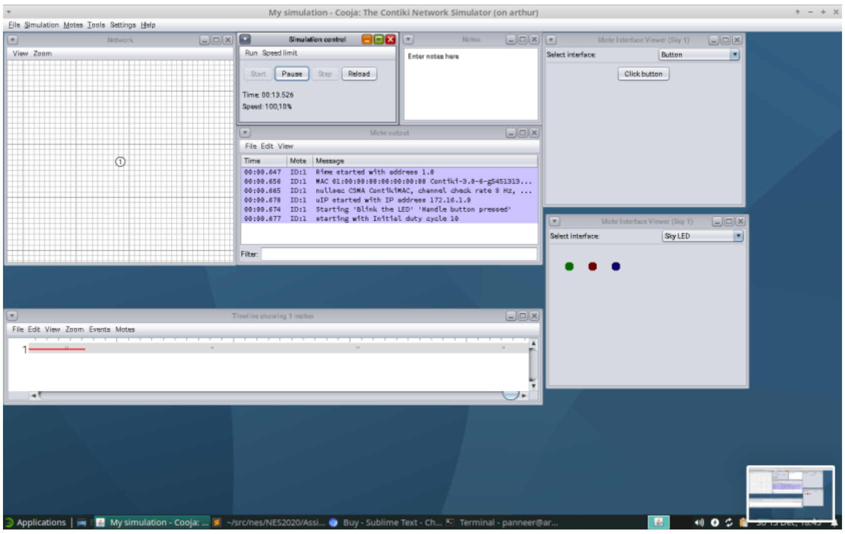
Task 2: Toggle red LED for unequal time intervals on time – 1s / off time – 0.5s

We extended the above implementation by adding a flag variable (i) to track the state of led(on/off) and based on the state of the led the timer is reset to new time using the etimer_set() function. Here, leds_on and leds_off is used to toggle the LED on and off.

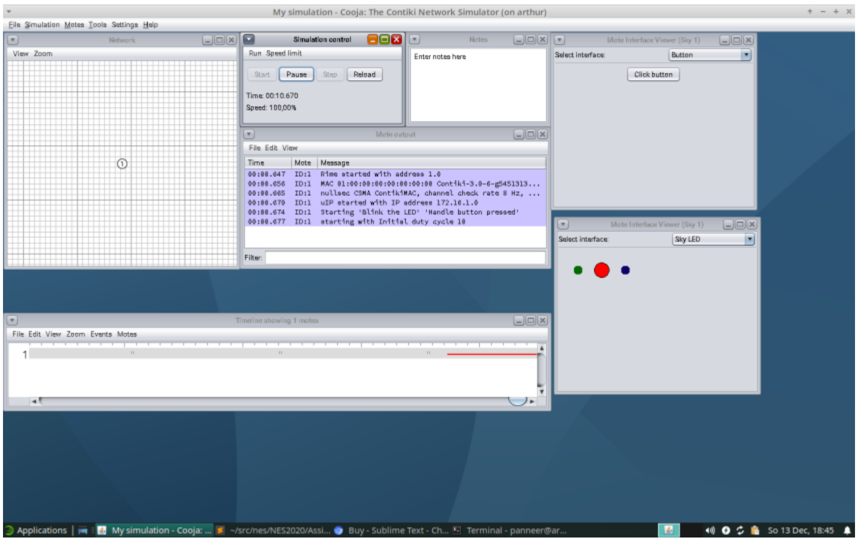
Protothread Implementation:

```
PROCESS_THREAD(led_pt, ev, data) {
    static struct etimer timer;
    static int i = 0;
    PROCESS_BEGIN();
    time_on = 1.0;
    time_off = 0.5;

    etimer_set(&timer, CLOCK_SECOND);

   while(1) {
     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
     if( i== 0 ){
         i = 1;
         leds_on(LEDS_RED);
         etimer_set(&timer, CLOCK_SECOND * time_on);
     }else if(i == 1){
        i = 0;
        leds_off(LEDS_RED);
        etimer_set(&timer, CLOCK_SECOND * time_off);
     }
   }

    PROCESS_END();
}
```

Output 1: led off



Output 2: led on



Task 3: Toggle the LED in different Brightness level (10% -> 50%->90%->10%..) using PWM on click of a button

We extended the above the implementation by adding one more protothread(btn_pt) which listen for the button press event. When the button is pressed the brightness level is toggled to next level(10 - > 50 -> 90 -> 10 ->50 -> ,....).

Code:

```
PROCESS_THREAD(btn_pt, ev, data) {
    PROCESS_BEGIN();

    SENSORS_ACTIVATE(button_sensor);

    while(1) {
        PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event && data == &button_sensor);
        duty_cycle = (duty_cycle == 90)? 10: (duty_cycle + 40);
        change_time();
    }

    PROCESS_END();
}
```

- Next, the on / off time for the brightness level is calculated dynamically in change_time( ) method based on the brightness level which is duty_cycle using the formula
    - *T= 1 / frequency;*
      period = time_on + time_off;
    - *duty_Cycle = (time_on / period) * 100;*
    - *time_on = (duty_Cycle * period) / 100;*
    - *time_off = period – time_on;*

code:

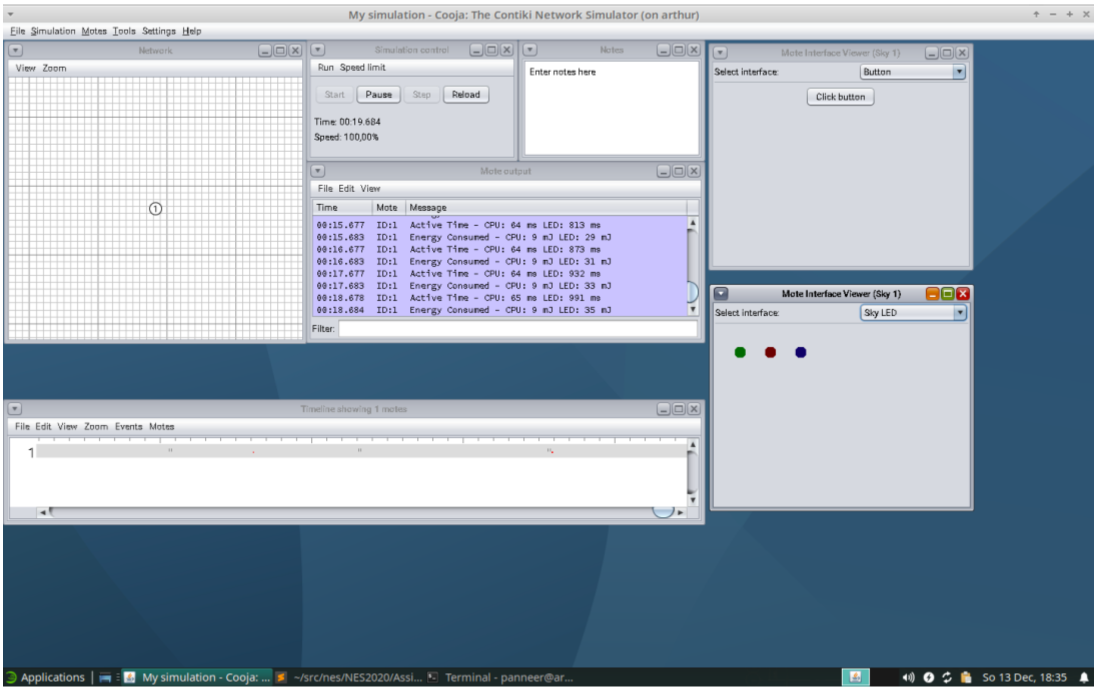
```
void change_time() {
    // Below formulas are  applied to calculate the time_on and time_off intervals
    // 1) Frequnecy is 50 hz
    // 2) Period ( T ) =  1 / f ;
    // 3) Period ( T ) = time_on + time_off;
    // 4) Duty_cycle = ( time_on / T ) * 100 ; => time_on = (T * period)/100 ;

    float frequency = 50;
    float period = 1 / frequency ;

    time_on = (float) (duty_cycle* period) / 100;
    time_off = (float)period - time_on;
}
```

- Since, time_on, time_off and duty_cycle variables are static. The updated values variables can be accessed from all the threads.
- Next, the on and off time of etimer which was implemented in the last task in the protothread(led_pt) will be updated for each button press and LED's will blink according to the duty cycle.

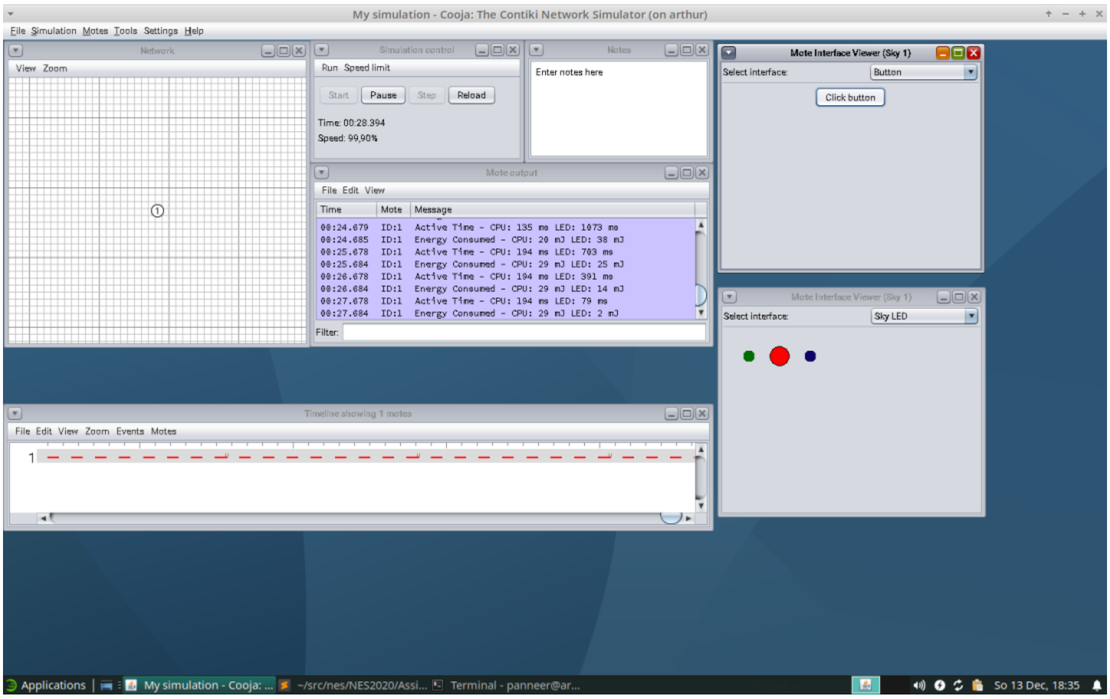
Code:

```
PROCESS_THREAD(led_pt, ev, data) {
    static struct etimer timer;
    static int i = 0;
    PROCESS_BEGIN();

    etimer_set(&timer, CLOCK_SECOND);

    printf("starting with Initial duty cycle %d\n", duty_cycle);

   while(1) {
     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
     if( i== 0 ){
         i = 1;
         leds_on(LEDS_RED);
         etimer_set(&timer, CLOCK_SECOND * time_on);
     }else if(i == 1){
        i = 0;
        leds_off(LEDS_RED);
        etimer_set(&timer, CLOCK_SECOND * time_off);
     }
   }

    PROCESS_END();
}
```

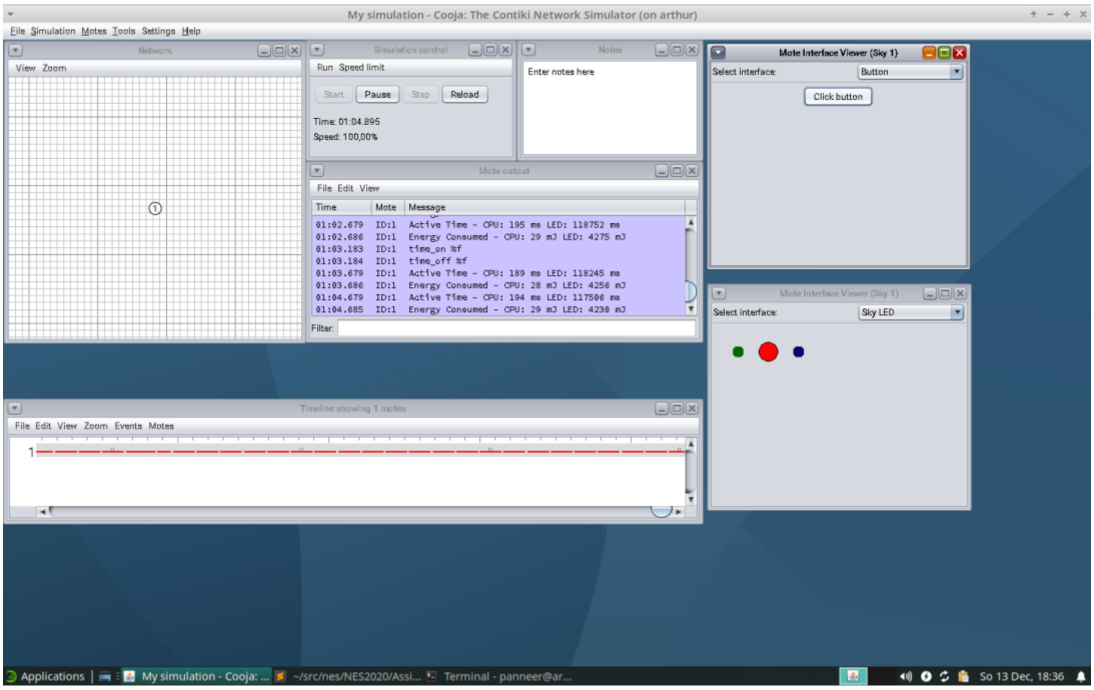
Output 1: 10%



Output 2: 50%



Output 3: 90%



<u>**Task 4:** Calculate and print the Energy consumption of CPU and LED for every second for the above implementation Using energest module</u>

We extended the above implementation by adding one more protothread(energy_pt) Which will be calulating the energy consumption as below.

- Intially, energest is initialized by #define ENERGEST_CONF_ON 1 in global config and energest_init() inside the protothread (energy_pt).
- Next, initial active time of CPU and LED is noted using the function energest_type_time() and the constants ENERGEST_TYPE_CPU for CPU active time and ENERGEST_TYPE_LED_RED for LED active time.
- A Timer using etimer is initialized for every second using the constant CLOCK_SECOND.
- When the timer expires, the new active time of CPU and LED is noted as described above.
- Now, To find the active time of CPU and LED in last second, The difference of new active time and initial active is calculated.
- This is converted to milliseconds by multiplying by 1000 and diving it by the constant RTIMER_SECOND because the energest module will provide the output in time ticks which needs to be converted to milliseconds.
- Energy consumption is calculated by multiplying the voltage , current and active_time of the component. Voltage and current for the component is available in the datasheet of the microcontroller. The energy consumption is printed in mJ and Time is printed in ms.
- Finally, The initial active time value is updated with the new active time to calculate the active time for next second.

Prothothread Implementation:

```c
PROCESS_THREAD(energy_pt, ev, data) {
    PROCESS_BEGIN();

    static unsigned long old_cpu_active_time;
    static unsigned long old_led_active_time;
    static struct etimer et;

    energest_init();
    // Initial active time of each component in ticks
    old_cpu_active_time = energest_type_time(ENERGEST_TYPE_CPU);
    old_led_active_time = energest_type_time(ENERGEST_TYPE_LED_RED);
    // Calculate enrgy consumption for every second
    etimer_set(&et, CLOCK_SECOND);
    /* Real-time clock */
    printf("RTIMER_SECOND: %u\n", RTIMER_SECOND);

    while(1) {
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

    energest_flush();
    //Total active time of each component untill current second in ticks
    unsigned long new_cpu_active_time = energest_type_time(ENERGEST_TYPE_CPU);
    unsigned long new_led_active_time = energest_type_time(ENERGEST_TYPE_LED_RED);
    //Active time of each component in last second in ms
    unsigned long cpu_active_time = (new_cpu_active_time - old_cpu_active_time) * 1000 / RTIMER_SECOND;
    unsigned long led_active_time = (old_led_active_time - new_led_active_time) * 1000 / RTIMER_SECOND;
    // Swap the values to calculate active time and energy consumption for next second
    old_cpu_active_time = new_cpu_active_time;
    old_led_active_time = new_cpu_active_time;
    printf("Active Time - CPU: %lu ms LED: %lu ms \n", cpu_active_time, led_active_time);
    // Energy consumption E = voltage * current * time
    // https://stackoverflow.com/questions/45644277/how-to-calculate-total-energy-consumption-using-cooja
    // Voltage and current values are from datasheet of the microcontroller
    unsigned long energy_consumption_cpu = 3 * 0.05 * cpu_active_time;
    unsigned long energy_consumption_led = 1.8 * 20 * led_active_time / 1000;
    printf("Energy Consumed - CPU: %lu mJ LED: %lu mJ \n", energy_consumption_cpu, energy_consumption_led);

    etimer_reset(&et);
    }

    PROCESS_END();
}
```

Output: