Networked Embedded Systems [NES]

Energy Consumption

20.01.2022

No mandatory submission!

Introduction

Building on the previous exercises, we will now investigate the impact of firmware on a mote's energy consumption. To do this, we will be using the *Energest* energy monitoring framework of Contiki, which can count how much time individual components, e.g., CPU, LEDs, transmitter, receiver, etc. of a mote have been active. As an example use case, we will be building firmware that uses Pulse-Width Modulation PWM (see fig. 1) to make a LED appear lit at different levels of its brightness.

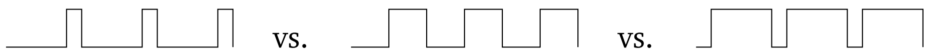

Figure 1: Pulse-Width Modulation (PWM)

1. Extract the given skeleton code from the ISIS course page.
2. First, we will investigate how to use *event timer* to generate timed events and build a firmware that toggles the red LED on the sensor mote every 1 s. The documentation of *event timer* is here[1].
   (a) Create a process that performs the following operations in an *endless loop*:
   (1) toggle the red LED on the sensor mote;
   (2) set an *event timer* to report a timed event after 1 s;
   (3) let the process wait for the expired event of the timer we have just set, then yield control to the operating system.
   Helpful functions: `PROCESS_WAIT_EVENT_UNTIL`, `etimer_set`, `etimer_expired`.
   Helpful constant: `CLOCK_SECOND`.
   (b) Run the firmware in the *Cooja* simulator and assert that the red LED is blinking.
3. Extend the process in item 2a that allows non-equal time intervals for the LED on and LED off durations, e.g., the LED is on for 1 s and off for 0.5 s. Global variables can be used to store the on/off intervals.

[1] see `https://github.com/contiki-os/contiki/wiki/Timers#The_Etimer_Library`

4. Next, we will learn how to use PWM to control the brightness of a LED. The idea is that if we turn the LED on and off fast enough, the viewer will see the LED with a constant light output which appears dimmer instead of flashing. The brightness of the LED is then decided by the duty-cycle (the total amount of time a pulse is ‘on’ over the duration of the cycle).

   We now extend the current firmware that can support making the red LED appear lit at 3 different levels of its brightness, namely 10 %, 50 %, and 90 % (see fig. 1).

   (a) Calculate the intervals for the LED on and LED off durations for each of these brightness levels.
   Assume that we use the oscillating speed of 50 Hz, or 50 times per second.

   (b) Create a second process that performs the following operations in an *endless loop*:

   (1) let the process wait for the event that the *user button* is pressed, then yield control to the operating system (*note*: the *user button* needs to be activated before using);

   (2) when the process resumes because the button was pressed, change the time intervals for LED on/off durations (e.g., 10 % → 50 % → 90 % → 10 % ...).

5. Finally, we now look at the impact of firmware on a mote’s energy consumption, using *Energest*[2] module.

   (a) Create a third process that uses *Energest* to collect (every second) how long the CPU and the red LED were active in the last second, expressed in time ticks (the default unit of Energest) and then outputs these two values.
   Helpful functions:

   - `energest_init(void)`: initialize the *Energest* module
   - `energest_flush(void)`: flush the time tick values for all components which are currently turned on
   - `unsigned long energest_type_time(int type)`: read the toal time tick value of a specific component type

   Helpful constants: `ENERGEST_TYPE_CPU`, `ENERGEST_TYPE_LED_RED`.

This concludes our exercises on “Energy Consumption”.

Contact

Gurjashan Pannu	<pannu@ccs-labs.org>
Course Website:	`https://isis.tu-berlin.de/course/view.php?id=25687`

[2]see `https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-Energest`