Wireless Networking

Course code: CS4222/5422, Assignment #2

Important Instructions: This assignment must be completed collaboratively by all members of the group. A statement of work detailing the contributions of each member to the assignment, along with the source code, must be uploaded to Canvas. Additionally, a brief video demonstrating the assignment is also required to be recorded and uploaded.

Objective of the assignment

This assignment is designed to facilitate the application of concepts learned in the lectures, focusing on sensors and actuators. You will become acquainted with programming sensors on the sensor tag platform. Your task involves programming the Contiki operating system to detect certain physical phenomena, processing the corresponding sensor data, and subsequently to initiate a response by activating the actuators on the platform. In the context of the Contiki OS, this assignment will involve learning about event scheduling, timer programming, and programming of sensors and actuators.

Introduction

The Texas Instruments Sensor Tag CC2650 is an example of a wireless embedded system or IoT platform, featuring ten sensors to measure light, sound, motion, magnetic fields, and temperature. This platform offers wireless networking capabilities through BLE and ZigBee standards. Designed for energy efficiency, sensor tag can operate for extended periods on small power sources such as coin cell batteries, albeit with constrained computational and memory resources on its microcontroller.

As a result, these embedded platforms do not run conventional operating systems such as Windows or Linux. Instead, these constrained IoT devices utilize operating systems like Contiki, an open-source system originally developed in Sweden by Adam Dunkels. Supported by a global community of developers, Contiki OS is designed for low-power microcontrollers, facilitating wireless networking and supporting various communication protocols and standards across a broad spectrum of platforms, including the SensorTag.

ContikiOS: https://github.com/contiki-ng/contiki-ng

Program execution within Contiki OS operates on an event-driven basis, with timers serving as one significant source of these events. Contiki offers a comprehensive suite of timer libraries for this purpose. In this assignment, you will explore the utilization of both the etimer and rtimer. Timers in Contiki

are managed using the **struct timer** data structure, and their operation is controlled through three specific function calls.

- **timer_set():** used to initialize and starts the expiration time.
- **timer_reset():** used to restart the timer from previous expire time.
- **timer_restart():** used to restart the timer from current time.

There are several differences between the two timers. Firstly, the difference stems from etimer and rtimer time resolution. etimer's clock resolution depends on the number of clock ticks per second (CLOCK_SECOND), while rtimer uses RTIMER_SECOND. Another difference is that programming style; Etimer uses a more "sequential" model while rtimer uses callbacks. Let us try to look at code involving these timers to understand these differences.

Sample code involving the use of etimer:

```
PROCESS_THREAD(example_process, ev, data)
{
PROCESS_BEGIN();
etimer_set(&timer_etimer, CLOCK_SECOND); /* Delay 1 second */
while(1) {
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer_etimer));
etimer_reset(&timer_etimer);
}
PROCESS_END();
}
```

Sample code involving the use of rtimer:

```
PROCESS_THREAD(process_rtimer, ev, data)
{
PROCESS_BEGIN();
init_opt_reading();
while(1) {
rtimer_set(&timer_rtimer, RTIMER_NOW() + RTIMER_SECOND, 0,
do_rtimer_timeout, NULL);
PROCESS_YIELD();
}
PROCESS_END();
}
```

You can find additional documentations regarding timers in the following link https://docs.contiki-ng.org/en/develop/doc/programming/Timers.html

Example Programs for performing Sensing, Actuation

We provide you three programs ton help you conduct the assignment.

- etimer-buzzer.c: a sample program that shows how to use the etimer and the buzzer
- rtimer-lightSensor.c: a sample program that shows how to read from the light sensor
- rtimer-IMUSensor.c: a sample program that shows how to read from the IMU sensor

Please download, understand, compile and then execute the above program. You can monitor some of the steps of the program by observing the console for the output generated using printf statements. Please do note that to compile these programs your makefile should include names of the program.

```
CONTIKI_PROJECT = etimer-buzzer rtimer-lightSensor
rtimer-IMUSensor
```

Specific tasks to be performed for the assignment

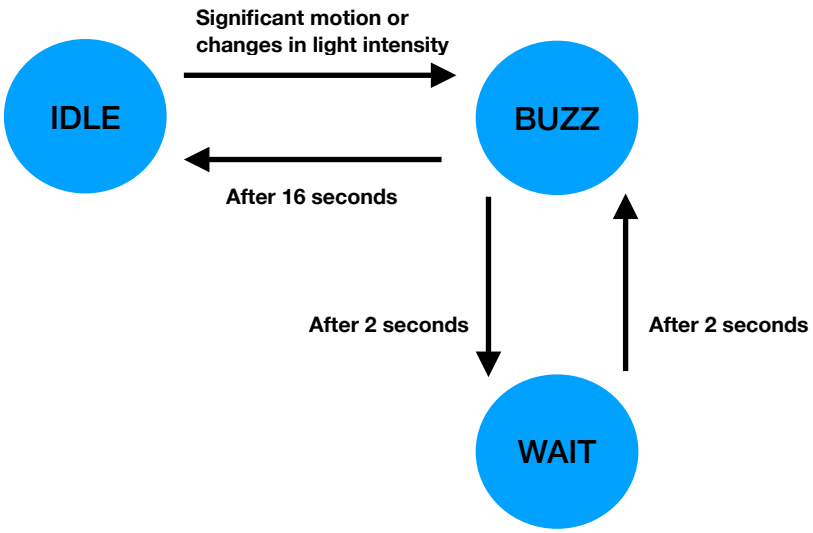
All the tasks are mandatory. Hence, make an effort to complete all of them.

Task 1: Finding the clock resolution for rtimer and etimer

Please program the device with the program etimer-buzzer. From the output of these program, please observe and note down the value of CLOCK_SECOND. Also please find out how many clock ticks corresponds to one second. Next, please program the devices with the rtimer-lightSensor and note down the value of RTIMER_SECOND from the output of the program. Please find out how many clock ticks corresponds to one second. Mention this as part of report to be submitted for the project.

Task 2: Buzzer actuation by tracking light and motion sensor

You next task is to interface light sensor and to actuate the buzzer.

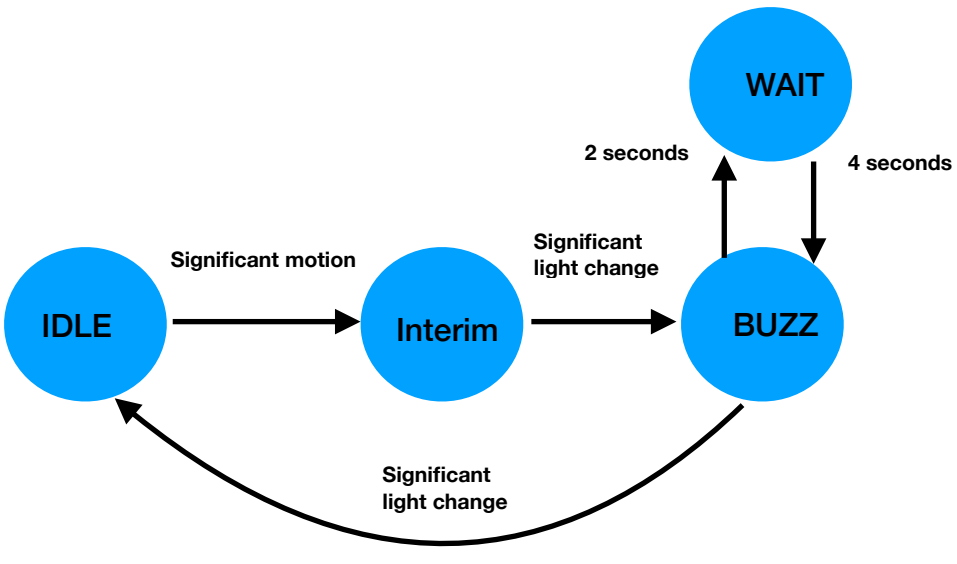


You will write a program to implement state machine similar to the one shown above. The program starts in the IDLE state. Upon detecting a substantial change in light or significant movement, it transitions to the BUZZ state. In this state, the sensor tag activates its buzzer to emit a sound for 2 seconds. Subsequently, the program enters the WAIT state, where it pauses for 2 seconds before reactivating the buzzer for another 2 seconds. This sequence is repeated several times. Finally, after a duration of 16 seconds from the entry into the BUZZ state, the program reverts to the IDLE state.

Please consider the following while writing the program:

- Significant change of light reading is can be defined by a change of more than 300 lux in intensity. Do not sample the light sensor at a rate higher than 4 Hertz as the driver does not work well
- Significant motion is defined as picking up or moving of your arm while holding the sensorTag in your hand. Do not sample the IMU sensor at a sampling rate higher than 50 Hertz as it may result in unstable behaviour

Task 3: Putting everything together to perform complex task



You will develop a sophisticated state machine for the program similar to the one shown above. The program starts with state machine in the IDLE mode. Upon sensing significant motion, it transitions to an INTERIM state. While in the INTERIM state, should there be a significant change in light intensity, the program shifts to the BUZZ state. In this state, the sensor tag triggers the buzzer, producing a sound for approximately 2 seconds. Following this, the program proceeds to the WAIT state, where it pauses for 4 seconds before reactivating the buzzer for an additional 2 seconds. This cycle of state transitions persists; however, upon detecting a significant change in light intensity once more, the program reverts to the IDLE state.

Demonstration, Submission Guidelines and Deadline

The deadline for submission of the assignment is **March 18, 2024.** This assignment needs to be done with members of the designated group.

Please submit a readme file, source code, statement of work and a video demonstrating the operation of the sensor tag. They need to be submitted onto the canvas portal. The statement of work should describe the contribution made by every member of the group to the assignment. You need to submit the source code for Task 2 and Task 3 as separate files. Only one member of the group should submit the assignment on the canvas.

There will be a penalty of 10% per day after 18th of March 2024.

The readme file should contain the following:

- Value of CLOCK_SECOND
- Number of clock ticks per second in 1s (real time) using etimer
- Value of RTIMER_SECOND
- Number of clock ticks per second in 1s (real time) using rtimer
- Instruction on how to execute your program
- Name and student ID of the members of the group