Software for Real-Time and Embedded Systems | B-KUL-H04L2a-2021

Project: Time-Synchronized Embedded Device

Arne Wouters, r0632965

November 22, 2020

Contents

1 The system

The system consists of 4 tasks:

- LoRaReceiver
- DatabaseManager
- LoRaSender
- CommandManager

There are 2 queues, the *DatabaseQueue* and the *LoRaSenderQueue*. We also have a semaphore that is required for every access to EEPROM.

1.1 LoRaReceiver

This task receives and reads incoming beacons. Packets with a size smaller than 5 will be ignored because we assume the message of the beacon has a length of atleast 5 characters. The system can handle packets with more characters as long as they follow the format, that the first 4 characters are the gateway ID and the remaining characters are the time until the next beacon transmission in seconds. Once a beacon is received, the time until the next beacon T is put in the *DatabaseQueue* and the *firstPackageReceived* flag is set to *true*. Next the *DatabaseManager* task is resumed and the task will go to sleep for T seconds. When the task processed 20 beacons, it will put the system in ultra low-power mode.

1.2 DatabaseManager

In the setup of the task, it takes the semaphore and reads the first 2 bytes of the EEPROM. These 2 bytes contain the address behind the last write in the EEPROM so it's the point where the task should start writing new values. In the loop, the task reads values from the *DatabaseQueue* and runs as long as there are items available in the queue. If the queue is empty, the task will suspend itself and only resume when the function *vTaskResume()* is called in the *LoRaReceiver*. The item in the queue gets dequeued. Next the task attempts to take the semaphore and waits for the semaphore if it isn't available. We get the temperature from the temperature sensor and then we write the temperature with the data from the queue to the database (EEPROM). After the write we update the address stored inside the program and the address stored in the first 2 bytes of the EEPROM. When the EEPROM is full, we reset and start writing from the first address again. After writing to the EEPROM, the semaphore is returned. The last step is putting the temperature into the *LoRaSenderQueue* and resuming the *LoRaSender* task. Now we have processed one item from the *DatabaseQueue* and we can start the loop again if more items are available in the queue.

1.3 LoRaSender

This task reads values from the *LoRaSenderQueue* and runs as long as there are items available in the queue. If the queue is empty, the task will suspend itself and only resume when the function *vTaskResume()* is called in the *DatabaseManager*. This task reads the temperature value from the queue, makes a packet with the temperature as message and sends it back to the gateway (GW).

1.4 CommandManager

This task reads commands from the serial port and prints output to the serial port at a rate of 9600 baud. There are 4 commands supported (the input for these functions is just the numbers 1, 2, 3 and 4):

1. Read the latest temperature value and beacon details from database and print the output to the serial port
2. Read all temperature values and beacon details from database and print the output to the serial port
3. Turn on ultra low-power mode
4. Reset the database

When the first package is received, the task removes itself as a task and disables the usb subsystem to save power.

2 Varying real-time constraint

The task *LoRaReceiver* will sleep for T seconds where T is the time in the received beacon. When the task wakes up again, it resumes the LoRa module (because it goes into sleep mode when the system is in low-power operation mode) and is ready again to accept beacons. T is not the duration that the whole system will be in low-power operation mode. The system only goes in low-power operation mode when all other tasks are finished. Usually the next beacon arrives 600-1000ms after the *LoRaReceiver* is ready. So it's possible to increase the duration of sleep a little bit for the task to minimize the power consumption but out of safety, we didn't to this here.

3 Synchronization

We use a semaphore for accessing the database. Only the *DatabaseManager* and the *CommandManager* acquire the semaphore. The semaphore always gets returned so a task will never get suspended or deleted before they release the semaphore.

4 Power consumption

4.1 Low-power operation mode

When there are no tasks active, the system goes into the *vApplicationIdleHook*. Here we turn off the ADC (Analog to Digital Converter), put the LoRa module in sleep mode and the microcontroller (MCU) in Idle mode. This leads to a power consumption of $\sim 2.90\,\text{mA}$. We leave the low-power operation mode as soon as the *LoRaReceiver* task gets active again.

4.2 Ultra low-power mode

After receiving 20 beacons or after executing the third command, the system goes into ultra low-power mode. First we remove the task scheduler and disable the LoRa module. Next we disable the usb subsystem (if the third command was used), turn off ADC and set all pins to output and low. We put the MCU in power down mode and use the function *power_ all_ disable()* to disable all modules. Here we have a power consumption of $\sim 814\,\mu\text{A}$.