Software for Real-time and Embedded Systems (B-KUL-H04L2A)

Report: SoRTES Project

Ruben Kindt (R0656495)

November 28, 2020

Contents

1 The varying real-time constraint

1.1 The varying real-time constraint: user input

Because we do not know when the user input will arrive, we should check often. We could check the serial receive buffer contiguously, but this would consume more power than checking it periodically. Checking periodically at a rate of one check per quarter second will be no different to the eyes of the user. We implemented this by creating a FreeRTOS task with a FreeRTOS delay of 250 ms by dividing 250 ms by the tick rate ‘portTICK_PERIOD_MS’.

We should have implemented the above with the use of interrupts. This would mean that when the user enters input the program would generate an interrupt and this would in turn execute a function call to process it. This way would most likely generate in the use of less power, but unfortunately, we were unable to implement this. Among others we looked like the ‘SerialEvent()’ function thinking it would generate an event when user input was given. But this function does not work on an interrupt level, It is called after each ‘loop()’ call.

1.2 The varying real-time constraint: gateway time until next transmission

Due to not knowing when the first beacon will arrive, we have the same situation as with the user input. We could listen continuously, but this would consume more power. Therefore, we only process the beacons at a rate of one per quarter second. We did this with the help of FreeRTOS delay of 250 ms as explained before. To be clear we do not disable the LoRa receiving during the FreeRTOS delay, because then the LoRa receive buffer would obviously not contain any received beacons.

After we received out first beacon, process and respond to it we can turn off our LoRa module because we know for how many seconds, we will not receive any beacons. Turning the module off will result in a current drop of around 16 mA. We turn off the ‘listeningForBeacons()’ task via a FreeRTOS delay of the time between beacons.

2 Task synchronization to ensure database consistency

Since we do not have a CPU with multiple cores, we cannot execute in true concurrency. To look like the task are concurrently FreeRTOS implements its scheduling like round robin. This means that a task can be put on ‘pause’ to continue another task. This could theoretically cause mayhem in the database. The chance of this happening is low, since the user has to match the user input exactly with the receival of a beacon and the task scheduler has to switch tasks right at the code where we read or write to the database. But because the chance of it happening is low does not mean it can not happen. The remove the possibility of it occurring is done by calling the ‘vTaskSuspendAll()’ function of FreeRTOS right before any changes to the database. This function of FreeRTOS prevents the scheduler from changing task, meaning that the LoRa write to database and the reads from user input cannot interfere with each other. After the database operation has finished, we do need to turn the task scheduler back on with the ‘xTaskResumeAll()’ function of FreeRTOS.

3 Low power consumption

3.1 Low power consumption: general notes

To reduce power usage in general we prevent our I/O pins from having a floating voltage, by defining them as output and pulling them to 0 Volts. For a comparison when we pull the I/O pins high we consume around 58 mA more, we expect this enormous increase of current usage to be a caused partially by the white LED on the board to be turned on. We also experimented with setting the pins as input, but this resulted in an increase of current by 0.1 mA.

The red LED probably uses a lot of power as well, but this is the power indicator led and cannot be turned off. The only way to turn it off is to desolder the connection.

3.2 Low power consumption: between consecutive transmissions

When the FreeRTOS tasks are not running, due to a FreeRTOS delay, the scheduler runs an idle task. Meaning that the CPU does some useless operations until one of the tasks need to be run again. This is not that good for the power usage. To solve this, we make use of the option within FreeRTOS to define our own idle function. Using the 'vApplicationIdleHook()' of FreeRTOS we change the idle function of FreeRTOS to a more power friendly sleep. Which reduced the current usage from 15.7 mA to 11.3 mA while in the 'ADC Noise Reduction' mode. The 'ADC Noise Reduction' mode compared to the 'Idle' mode it consumes 0.7 mA less current. We also experimented with the 'Power-save' mode, the 'standby' mode and the 'Extended standby' mode but we could not get a reliable acknowledge back to the gateway in those modes. To sleep we first set the mode using 'set_sleep_mode' to the 'ADC Noise Reduction' mode, then disable the interrupts to execute the following function without interrupts, set sleep to be enabled, then reenable the interrupts and finally going to sleep via the 'sleep_cpu()' function. We do believe that the 'Power-save' mode would be the best to implement here since we only need to wake up after a sleep of around two to nine seconds.

3.3

Low power consumption: ultra low-power mode For the ultra-low-power mode implementation we make sure to turn as mush as possible of. We delete the possible running tasks via the 'vTaskDelete()' function of FreeRTOS, we end the Serial port and the LoRa module, we reset our pins to be output and low, we disable the ADC and finally we go to the 'Power down' mode. We also disable brown-out detection by changing the extended fuses of the board.txt file that is used to flash the PCB to the value 0xf7 which disables the brown out detection. All the above settings results in a current usage of 11.3 mA, the same current usage as in the between the transmissions which leads us to conclude that we forgot to disable some core components of the PCB.

To find this last components we tried al lot of things, including a 'power_all_diasble()' function of the avr/pwer.h library which weirdly increased the power consumption.