Autonomous Car

Introduction

Real time operating systems are composed of various tasks. Tasks are infinite loop functions that serve certain functionalities. The operating system Kernel switches the processor resources (processing time, hardware peripherals, etc.) between tasks. The goal of this project is to implement a simple scheduler that switches between various tasks non preemptively.

1 Description

A scheduler is the core of the **RTOS**. You will create a non-preemptive scheduler that schedules different tasks and call them whenever their periodicities fire. You have to drive a light tracker car that's equipped by:

1. Two Photo-Resistors.
2. One Ultrasonic Sensor Module HC-SR04.
3. On-board Temperature sensor
4. One LCD.
5. Four motors + Two motor drivers.

- The car motors are started using one of the two on-board switches.
- The car will stop if the other switch is pressed or a time of one minute is elapsed.
- When motors are started, the car moves and swings to the way with the highest illumination.
- The Temperature, the LDR.s difference, and the elapsed time are displayed on the LCD.
- If the car gets closer to an obstacle by **max 10 centimeters**. The ultrasonic sensor fires and the car reverses its direction first, moving to the back then rotates by **90 degrees**.

The scheduler will be called in the main and will schedule according to **a counter that is incremented in the Systick handler**. A pseudo code for the main function is:

```
int main(void){

    1- Initialize and setup the Hardware.

    2- Create different tasks through a function as below:
       /*create_task (A pointer to the periodic task created , Task periodicity)*/
       void create_task (void (*task)(), uint32_t ms_periodicity )

    3- Start the scheduler through:
       void tasks_scheduler (void)

    /* this Loop will never be reached */

    While (1);

}
```

1.1 Scheduler

Use the SysTick timer to create a simple non-preemptive scheduler. The scheduler implementation must be in separate files (e.g., **scheduler.c** and **scheduler.h**). the Scheduler supports multiple tasks using a macro "**NUM_OF_TASKS**". scheduler files have:

1- a struct that acts as a task control block. The struct has two members (pointer to task and task period).
2- The function used to create tasks:
   void **create_task** (void (*Task)(), uint32_t ms_periodicity)
3- The function used to schedule the tasks:
   void **tasks_scheduler**(void)
   /*This task has a while (1) loop that scans the periodicities inside*/

1.2 Tasks

At the very beginning, Tasks **should not** have a while(1) inside. You have mainly three tasks to implement:

1- Task to swing the car based on the LDR.s [**void ldr_swing_car(void)**]
2- Task to display on the LCD. [**void lcd_display(void)**]
3- Task to avoid obstacles based on the readings of the ultrasonic sensor.
   [**void avoid_obstacles(void)**]

1.3 Project Levels

The project must be divided into many layers as shown below. Lower layers can't call the upper layers, but the opposite can happen. You must stick to the names of the files provided here.

<table>
<tr><td colspan="6">Tasks and the main function</td></tr>
<tr><td colspan="6">Scheduler.c/.h</td></tr>
<tr><td>Temperature.c/.h</td><td>LCD</td><td>Motors</td><td>LDR</td><td>Ultrasonic</td><td rowspan="2">General functions</td></tr>
<tr><td>GPIO.c/.h</td><td>GPTM</td><td>SysTick</td><td>ADC</td><td>PWM</td></tr>
</table>

2 Requirements

Project must be built as described above. All peripherals must be configured using the register level, **TivaWare** is not allowed. You must follow the provided names of files, and tasks.