Progetto di Sistemi Embedded e Real Time – Anno di corso 2020/2021

Federico Bertoli Matricola: 157896

Nome progetto: RTES-GrandPrix

Repository Github: [thebertozz/RTES-GrandPrix]

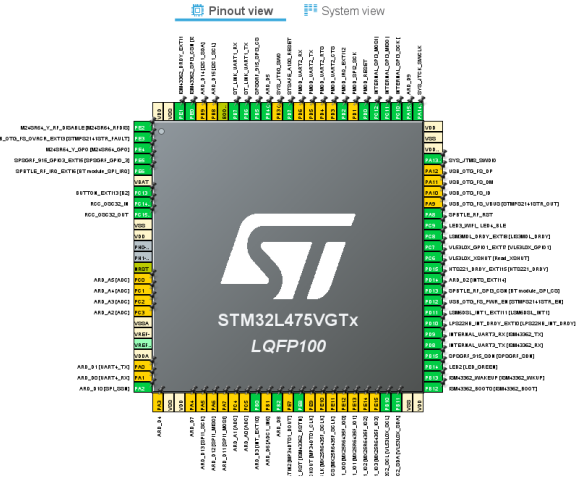
Introduzione

Essendo un grande appassionato di automobilismo e di giochi di corsa, ho pensato di ricreare una sorta di mini gioco di F1 in stile game boy da poter utilizzare sulla board, idealmente con uno schermo LCD connesso.

Sono stati usati i seguenti componenti della scheda per soddisfare i requisiti richiesti:

- **Inputs**: USER button, Sensore di prossimità (VL53L0X), Accelerometro (LSM6DSL)
- **Outputs**: Scrittura su seriale, LED
- **Sensori**: Temperatura (HTS221), Umidità (HTS221) e Pressione (LPS22HB)

Il progetto è stato creato con CubeMX per creare il pinout di default della scheda, per poi importarlo all'interno di CubeIDE.



1 Il pinout della scheda

Funzionamento

Il programma si avvia, richiedendo all'utente la pressione del tasto USER per avviare la gara; nel mentre, vengono stampate a schermo le informazioni relative alla "pista", quali temperatura, umidità e pressione:

```
Track pressure update: 1016 mBar
Track temperature update: 22.00 C
Track humidity update: 44 %

Press the USER button to start the Grand Prix...
```

Alla pressione del tasto USER da parte dell'utente, la gara viene avviata e viene attivato il LED per mostrare il semaforo verde, l'accelerometro per calcolare la posizione dell'auto in pista e la seriale per la stampa a schermo. Vengono inoltre stampati gli "avversari", che l'utente dovrà evitare muovendo la scheda simulando un volante.

```
Green light!
|           |
|           |
|           |
|           |
|         * |
|           |
|     x     |
```

Alla pressione del tasto USER durante la gara, verrà simulato un PIT STOP, per il quale viene utilizzata l'informazione del sensore di prossimità per capire il momento in cui sarà terminato.

```
------- PIT STOP -------

Keep your hands on the wheel!
```

Se l'utente non mantiene la mano sulla scheda (e dunque sul sensore di prossimità), il Pit stop non potrà essere completato.

```
Hold steady...!

|               ||[   ]|
|               ||  x  |
|               ||[   ]|
```

Mantenendo invece la mano sulla scheda, verrà mostrata a schermo l'informazione relativa alla sosta in corso.

Al termine, viene mostrata l'indicazione di ripartenza e la gara riprende.

```
GO GO GO!
```

La gara termina dopo 150 esecuzioni del task denominato raceDataPrintTask, viene mostrato un messaggio di fine corsa e il programma ritorna in attesa dell'input utente:

```
Chequered flag, good job!
```

È inoltre possibile fermare la gara prima del tempo premendo il tasto USER durante un Pit stop, per ritirare la macchina e ritornare in attesa dell'input utente per una nuova corsa:

```
We need to retire the car! Sorry.
```

Implementazione

Viene utilizzata una struct di gestione del programma:

```
struct manager_t {

    int temperature_value;  // Shared measured temperature value
    uint8_t humidity_value; // Shared measured humidity value
    int pressure_value;  // Shared measured pressure value
    BSP_MOTION_SENSOR_Axes_t  accelerometer_value; //Shared accelerometer value
    uint16_t proximity; //Shared proximity value
    uint8_t status;
    uint8_t b_green_light, b_track_data, b_user_button, b_temperature, b_humidity, b_pressure, b_proximity, b_race_data, b_accelerometer;
    uint8_t pit_stop_executions;
    uint8_t waiting_for_race_director_executions;
    uint8_t race_executions;
    uint8_t opponent_executions;
    int last_opponent_value;

} manager;
```

Questa struct è acceduta in mutua esclusione tramite un mutex apposito:

```
osMutexId managerMutexHandle;
```

È inoltre presente un semaforo binario, che viene utilizzato per gestire l'input dello USER button, intercettato tramite l'apposita callback:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

    if (GPIO_Pin == USER_BUTTON_PIN) {

        osSemaphoreRelease(userButtonInterruptSemaphoreHandle);
    }
}
```

Questo sblocca il task apposito di gestione, che avviandosi sarà in attesa di poter ottenere il semaforo per effettuare le operazioni.

Sono stati inseriti i seguenti task, con quelli relativi alle informazioni di ambiente (temperatura, umidità e pressione) con priorità al di sotto di quella nominale:

```
void startGreenLightTask(void const * argument);
void startTrackDataPrintTask(void const * argument);
void startUserButtonTask(void const * argument);
void startProximitySensorTask(void const * argument);
void startRaceDataPrintTask(void const * argument);
void startAccelerometerTask(void const * argument);
void startTemperatureSensorTask(void const * argument);
void startHumiditySensorTask(void const * argument);
void startPressureSensorTask(void const * argument);
void startButtonInterruptTask(void const * argument);
```

Per quanto riguarda l'ottimizzazione delle risorse, è stata utilizzata l'API **uxTaskGetStackHighWaterMark** per verificare lo stack ancora disponibile per i singoli task; partendo da una dimensione di 1024 words per ognuno, ho ottenuto valori via via inferiori arrivando vicino alle 40/50, che indicano che il task sta utilizzando quasi tutte le risorse messe a disposizione.

```
user button task watermark 908
temperature task watermark 111
humidity task watermark 184
pressure task watermark 134
green light task watermark 107
track data task watermark 884
proximity sensor task watermark 196
race data task watermark 122
accelerometer task watermark 158
```

2Una fase intermedia

```
green light task watermark 37
track data task watermark 40
user button task watermark 12
proximity sensor task watermark 58
race data task watermark 42
accelerometer task watermark 38
humidity task watermark 17
pressure task watermark 42
temperature task watermark 42
```

3Il risultato della fase di ottimizzazione

Sono stati inoltre utilizzati degli uint8_t dove possibile, per minimizzare l'utilizzo della memoria.