Ihssane JELLAL
Haythem BOUGHARDAIN
Sacha MEURICE

ELEC-H-410
Pandemic, the RTOS game

2022-05-19

Introduction

The aim of this project is to beat a pandemic game using FreeRTOS tasks management. The game simulates a real-time pandemic situation where the population decreases as the contaminations grow. To defeat the game, the objective is to produce as fast as possible enough vaccines before the population's counter reaches 0%. If this counter reaches 0%, the game is over.

The project is composed of one research lab. The lab can either work on a vaccine, or produce medicines. The difficulty in this game is that the lab cannot work on both tasks at the same time. Furthermore, once the lab is working on a task, it sould work on it until completion.

1 Code Design

The code in main.c is composed of four main tasks (order by priority):

- ☐ quarTask() : Quarantine the population when a contamination is released ;
- ☐ vaccTask() : Try to produce a vaccine when a clue is released ;
- ☐ medTask() : Produce pills when nothing else is being done ;
- ☐ lcdcTask() : Manage the display on the LCD screen.

1.1 quarTask()

This task is responsible to quarantine the population when a new contamination is released by the game. It checks every 6 ms a boolean flag `quar_flag` indicating if a contamination occurs. A signal is sent on the GPIO ports during the execution time of the `quanrantine`() function. It resets the flag when the quarantine is done.

This task has a priority of 19. It is the highest priority task after `gameTask`, as the population should be placed in quarantine within 10ms after a contamination.

```c
void quarTask()
{
    for (;;) {
        vTaskDelay(QUAR_PERIOD);

        if (quar_flag) {
            GPIOJ14_Write(1);
            quarantine();
            GPIOJ14_Write(0);

            quar_flag = 0;
        }
    }
}
```

The `quar_flag` is set to 1 as follow:

```c
void releaseContamination(void) {
    quar_flag = 1;
}
```

1.2 vaccTask()

This task is waiting to receive a message in the global queue called clueQueue. When it receives a new message from the queue, it checks if it has enough remaining time to produce the vaccine. As a vaccine takes 2,5 s to be produced, and the vaccine should be returned in 3 s, the delay cannot be higher than 500ms.

If the task can produce the vaccine on time, it waits for the lab_mutex resource. This resource is shared with the task producing pills. The lab_mutex is a semaphore that allows mutual exclusion between vaccTask() and medTask(). It prevents the lab to work on both tasks in parallel.

This task has a priority of 18. Indeed, if a clue is released, this task should be executed prior to the one producing pills.

```c
void vaccTask()
{
    Token clue;
    Token result;

    for (;;) {

        xQueueReceive(clueQueue, &clue, portMAX_DELAY);

        if (xTaskGetTickCount() - time_cnt < 500) {
            xSemaphoreTake(lab_mutex, portMAX_DELAY);

            GPIOJ12_Write(1);
            result = assignMissionToLab(clue);
            GPIOJ12_Write(0);

            xSemaphoreGive(lab_mutex);
            shipVaccine(result);
        }
    }
}
```

The clueQueue receives new messages through the releaseClue function:

```c
void releaseClue(Token clue) {

    time_cnt = xTaskGetTickCount();

    GPIOJ11_Write(1);
    xQueueSend(clueQueue, &clue, portMAX_DELAY);
    GPIOJ11_Write(0);
}
```

1.3 medTask()

This task first waits for the lab_mutex to be available before executing its instructions. In the context of the game, this means that before producing medicines, the lab should be free and not producing vaccines for example, as mentioned above in vaccTask(). Indeed, the goal is to produce medicine when nothing else is being done. When lab_mutex is available, medTask() calls the assignMissionToLab() function with a zero parameter. When this is done, the medicine can then be shipped.

This task has a priority of 17 as producing medicines is less important than producing vaccines.

```c
void medTask()
{
    Token result;

    for (;;) {
        xSemaphoreTake(lab_mutex, portMAX_DELAY);

        GPIOJ13_Write(1);
        result = assignMissionToLab(0);
        GPIOJ13_Write(0);

        xSemaphoreGive(lab_mutex);
        shipMedicine(result);
        vTaskDelay(MED_PERIOD);
    }
}
```

1.4 lcdTask()

This task displays each 200 ms the results of the percentage of the population, the number of vaccines produced and the number of medicines produced. It is the task with the lowest priority because the display of the different results is the least important task in this game, it shouldn't interfere too much with other tasks.

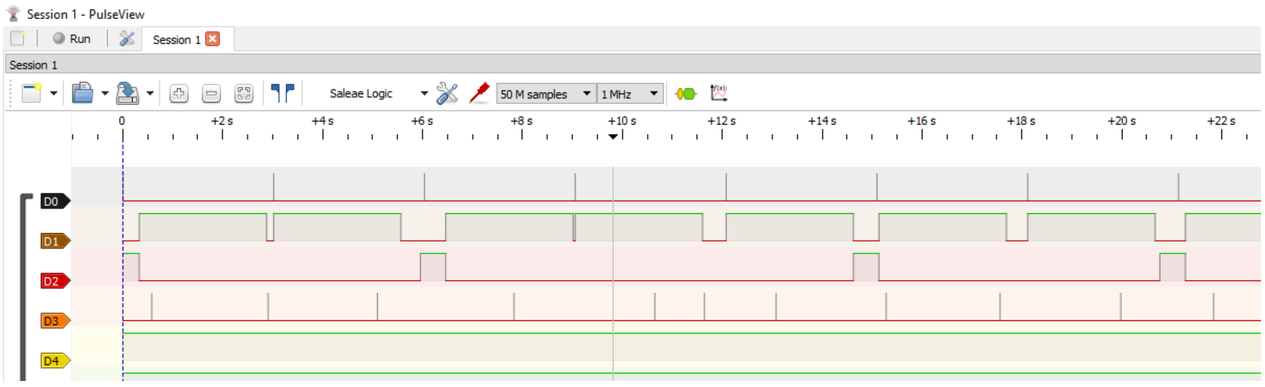
```
void lcdTask()
{
    for (;;)
    {
        LCD_ClearDisplay();

        LCD_PrintNumber(getPopulationCntr());
        LCD_PutChar(' ');
        LCD_PrintNumber(getVaccineCntr());
        LCD_PutChar(' ');
        LCD_PrintNumber(getMedicineCntr());

        vTaskDelay(LCD_PERIOD);
    }
}
```

2 Logic analyzer traces

The logic analyser is an electronic device able to measure and display signals coming from a digital system. It is used to keep traces of what happens in the system in real-time. In this project, the logic analyzer keeps traces of signals on four channels (D0, D1, D2 and D3), the first one is used to trigger the capture. The table below shows which event is tracked on which channel:

Channel	Event tracked
D0	A new clue is released by the game
D1	The lab is working on vaccines
D2	The lab is producing pills
D3	New random contamination

2.1 Real-time capture



In the figure above, the channel D1 is high when a new clue is released and the channel D2 is low: indeed, the lab cannot work on both medecines and vaccines at the same time. Each task must be completed before another one can be done.

The channel D3 shows when a random contamination occurs. The duration time of quarantine the population is quite low and this task takes the priority other the other ones as defined in the code.

For example, at 6 seconds, a new clue is released by the game. The lab is already working at this time on producing pills (D2 high). It is only when the pills have been produced, that D2 goes down and enables D1 high.

3 Conclusion

Ultimately, the written code was programmed into the provided PSoC to test the course of the RTOS game: results showed that the game was won by reaching 102 vaccines, with a final population of 17 people. This shows that our implementation managed to reach the expected objectives of researching for vaccines, producing medicines, and quarantining the population when outbreaks occurred. This was done by using tasks in FreeRTOS and ensuring real-time responses were efficiently and optimally handled, thanks to inter-task communication, resource sharing, synchronisation and other studied concepts.