EJECUTIVO CÍCLICO

La aplicación desarrollada para el reto de la asignatura cuenta con las siguientes tareas asociadas a las siguientes máquinas de estados:

- BUFFER → 518 Ticks → 6,47 µs
- BATTERY → 384 Ticks → 4,8 µs
- RENT → 384 Ticks → 4,8 µs
- DIGIT → 93 Ticks → 1,16 µs
- UPDATE → 22339 Ticks → 279,23 µs
- SEND → 216 Ticks → 2,7 µs

Tenemos en cuanta para la planificación los siguientes factores:

- Los tiempos han sido medidos con la función asm_ccount().
- El ESP8266 tiene un reloj de 80 MHz, por lo que 1 Tick = 12,5 ns.
- Como todas las tareas son muy inferiores al TICK_RATE configurado de RTOS, se aproximan todos los costes a 1ms.

Todos estos planteamientos quedan resumidos en la siguiente tabla:

Tarea	Costes (ms)	Periodo (ms)
T_1	1	1000
T_2	1	200
T_3	1	200
T_4	1	100
T_5	1	500
T_6	1	1000

Siendo en la misma:

T_1 ≡ Tarea correspondiente a la máquina de estados del buffer.

T_2 ≡ Tarea correspondiente a la máquina de estados de la batería.

T_3 ≡ Tarea correspondiente a la máquina de estados del alquiler.

T_4 ≡ Tarea correspondiente a la máquina de estados para los dígitos.

T_5 ≡ Tarea correspondiente a la máquina de estados para los indicadores led.

T_6 ≡ Tarea correspondiente a la máquina de estados para los envíos.

De esta manera, fijamos en primer lugar el hiperperiodo del sistema como:

M.C.M (100,200,500,1000) → 1000

A continuación, calculamos el periodo de los ciclos secundarios como:

M.C.D (100,200,500,1000) → 100

Una vez realizados estos cálculos pasamos a la colocación de las tareas en el cronograma, comenzando por los de periodo más bajo, como se muestra en la siguiente figura:

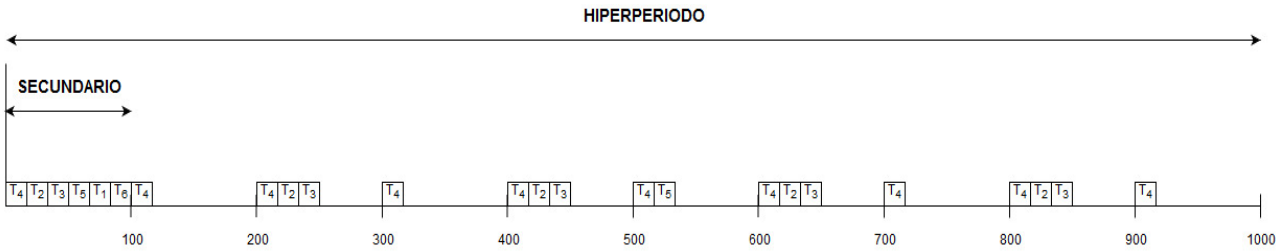

Figura 1: Cronograma del ejecutivo cíclico.

La implementación software en C que se ha llevado a cabo es la siguiente:

```
while(true) {
  switch (frame){
    case 0:
      fsm_fire(fsm_digit);
      fsm_fire(fsm_battery);
      fsm_fire(fsm_rent);
      fsm_fire(fsm_update);
      fsm_fire(fsm_buffer);
      fsm_fire(fsm_send);
      break;
    case 100:
    case 300:
    case 700:
    case 900:
      fsm_fire(fsm_digit);
      break;
    case 200:
    case 400:
    case 600:
    case 800:
      fsm_fire(fsm_digit);
      fsm_fire(fsm_battery);
      fsm_fire(fsm_rent);
      break;
    case 500:
      fsm_fire(fsm_digit);
      fsm_fire(fsm_update);
      break;
    default:
      break;
  }
  frame = (SECONDARY + frame) % (HYPERPERIOD);
  vTaskDelayUntil(&xLastWakeTime, SECONDARY/portTICK_RATE_MS);

}
```