Especificaciones

El objetivo del sistema es la supervisión automática de recursos mediante dos tipos de módulo hardware. La autenticación del acceso a recursos se realiza mediante tarjetas RFID, y la comunicación entre los módulos se realiza por Bluetooth o UART en comunicaciones *master-slave*. Para la gestión de usuarios y *log* de acceso, se utiliza una base de datos proporcionada por Microsoft Azure.

Uno de los módulos hardware es el *hub*, encargado de gestionar los permisos de acceso y el *log*, y de abrir la puerta del aula. Este se implementa en una Raspberry Pi 3, utilizando Raspbian. El otro módulo es el *locker*, encargado de controlar el acceso a una taquilla del aula; se utiliza uno de estos módulos por cada taquilla. Además del control de acceso, estos módulos harán saltar una alarma cuando permanezcan abiertas más del tiempo estipulado o cuando se abran sin permiso. Son implementadas en STM32F4.

Modelo de máquinas de estados

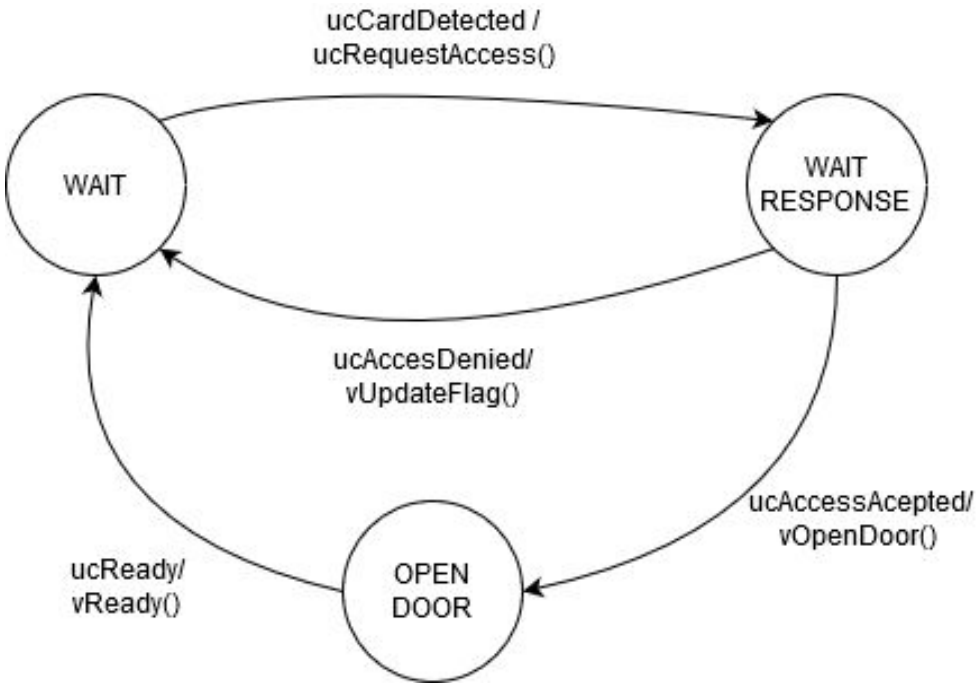

Figura 1: Diagrama de FSM_Locker

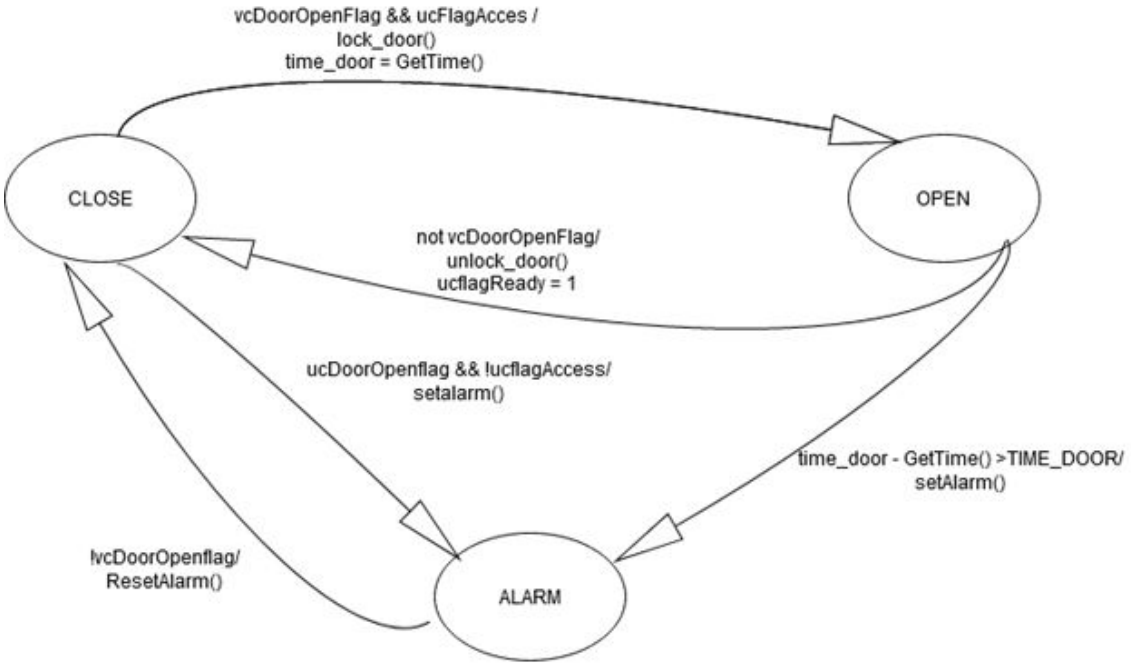

Figura 2: Diagrama de FSM_Alarm

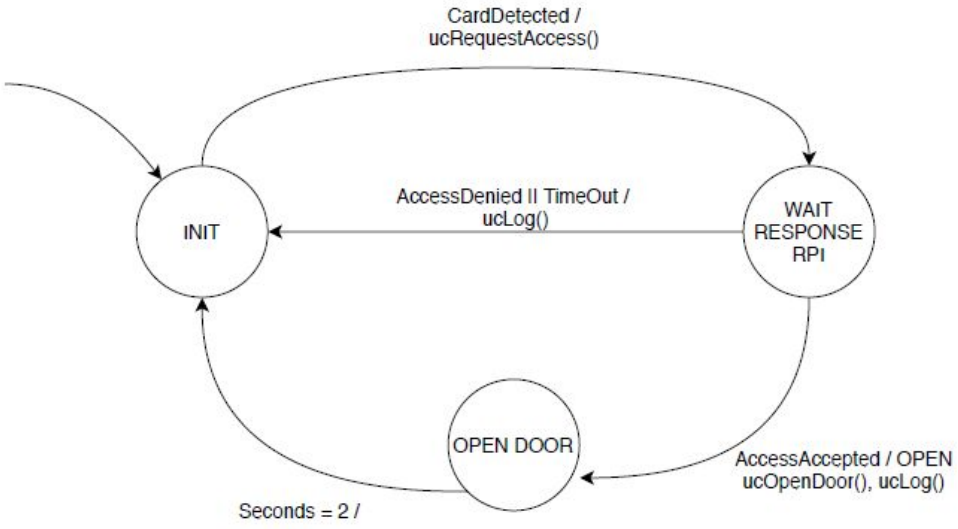

Figura 3: Diagrama de FSM_Hub

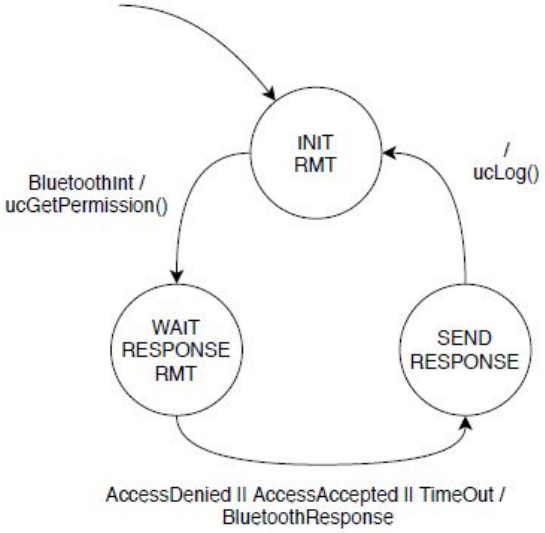

Figura 3: Diagrama de FSM_RMT

SPIN

Se utiliza un *active proctype* distinto para representar cada una de las cuatro FSMs. De la misma manera, se utiliza otro para simular la interacción con el exterior, y otros cuatro simulando el API y los *timeouts* de las FSMs.

Descripción de las siguientes propiedades en LTL para Spin:

- **“La FSM del Locker siempre vuelve al estado inicial”**
  ltl LockerNoBlock { [](<>(LockerState == INIT_locker)) }

- **“La FSM del Hub siempre vuelve al estado inicial”**
  ltl HubNoBlock { [](<>(HubState == INIT_hub)) }

- **“La FSM de Remoto siempre vuelve al estado inicial”**
  ltl RMTNoBlock { [](<>(RMTState == INIT_RMT)) }

- **“Un Timeout en la FSM de Locker no se activa si otra ya está activo”**
  ltl LockerTimeout { [](!(StartTimeoutLocker && TimeoutLocker)) }

- **“Un Timeout en la FSM de Hub no se activa si otra ya está activo”**
  ltl HubTimeout { [](!(StartTimeoutHub && TimeoutHub)) }

- **“Un Timeout en la FSM de Remoto no se activa si otra ya está activo”**
  ltl RMTTimeout { [](!(StartTimeoutRMT && TimeoutRMT)) }

- **“Si se ha escrito un log, no se borra”**
  ltl NoLogProblems { [](Log > 0) -> [](Log > 0)) }

- **“El Hub no procesa un petición del Locker si no se ha detectado una tarjeta”**
  ltl LockerToHub { !(RMTState == WAIT_RESPONSE_RMT) W CardDetectedLocker }

- **“La puerta del Locker no se abre si no se le ha dado permiso por el Hub”**
  ltl OpenLockerDoor { !(LockerState == OPEN_DOOR_locker) W PermissionAcceptedRMT }

- **“El Hub no procesa un petición de él mismo si no se ha detectado una tarjeta”**
  ltl HubToHub { !(HubState == WAIT_RESPONSE_RPI_hub) W CardDetectedHub }

- **“La puerta del Hub no se abre si no se le ha dado permiso”**
  ltl OpenHubDoor { !(HubState == OPEN_DOOR_hub) W AccessAcceptedHub }

- **“No se puede aceptar y denegar el permiso del Locker al mismo tiempo”**
  ltl AccessLocker { [](!(AccessAcceptedLocker && AccessDeniedLocker)) }

- **“No se puede aceptar y denegar el permiso del Hub al mismo tiempo”**
  ltl AccessHub { [](!(AccessAcceptedHub && AccessDeniedHub)) }

- **“No se puede aceptar y denegar el permiso remoto del Locker al mismo tiempo”**
  ltl PermissionRMT { [](!(PermissionAcceptedRMT && PermissionDeniedRMT)) }

- **“La alarma no se enciende si la puerta no está abierta o el timeout no se ha activado”**
  ltl AlarmStart { !(AlarmState == ALARM_alarm) W (DoorOpenedAlarm || TimeoutAlarm) }

Decisiones de implementación

En el Locker se decidió utilizar dos máquinas de estados separadas, una encargada de la interacción con el usuario y otra encargada de vigilar el estado de la puerta para controlar posibles aperturas sin autorización u olvidos por parte del usuario.

Se utilizan interrupciones para poder interactuar con la tarjeta del usuario para realizar la lectura de su rfid, y para que al pulsar el botón de usuario se simule el sensor de puerta abierta

o cerrada.

La implementación en el Locker se ha realizado utilizando un ejecutivo cíclico, se ha tomado esta decisión ya que se tratan de dos máquinas de estados sencillas que no bloquean nunca la una a la otra. Se ha medido los tiempos de ejecución en caso peor para poder planificar y después optimizar el algoritmo mediante YDS.

En el HUB se ha dividido las tareas de comunicaciones con el locker en una máquina de estados y de autenticación en la otra. Evitando que la autenticación contra el servidor de Azure bloquee las comunicaciones con el locker. Además el separar la autenticación se permite una mayor modularidad que permitiría, en un futuro, almacenar una base de datos local coherente con el servidor de Azure que se actualice cada día para reducir la latencia en los accesos.

Justificación de la validez de la verificación:

Al realizar la implementación mediante los ficheros fsm.h/.c, se garantiza que es una implementación directa de los modelos de FSM y así, la verificación realizada con SPIN es equivalente y por tanto garantiza el funcionamiento de lo implementado.

Optimización de consumo

Para la optimización de consumo se ha decidido implementar YDS para un ejecutivo cíclico del sistema del locker. Por tanto, las dos máquinas de estados implementadas son FSM_Locker y FSM_Alarma que se ejecutan periódicamente para garantizar que dan una respuesta adecuada al usuario.

Descripción del funcionamiento del sistema

El sistema empotrado tiene la FSM_Locker que se encarga de interactuar con el usuario y dar o denegar el acceso al recurso. Por otro lado se tiene la alarma que se encarga de si se abre la puerta sin autorización o se deja abierta notifique al usuario.

Se ha priorizado la seguridad frente a la responsividad del sistema por tanto la FSM_Alarma es la más prioritaria y tiene un plazo de 50 ms frente a los 300 ms que tiene la tarea de Locker.
Los tiempos de ejecución se han medido con el micro a 84 MHz.

	C	T	D
Locker	189 ms	300 ms	300 ms
Alarma	13 ms	50 ms	50 ms

Se ha seleccionado un hiperperiodo de 300 ms y se ha pasado a calcular la intensidad de cada intervalo.

$Hiperperiodo =$ 300 ms

$$G[z, z'] = \sum_{v_i \varepsilon V'([z,z'])} C_i / (z' - z)$$

$$G[0,\ 50] = \frac{C_{alarma}}{50\ ms} = \frac{13\ ms}{50\ ms} = 0.26$$

$$G[0,\ 300] = \frac{6 \cdot C_{alarma} + C_{locker}}{300\ ms} = \frac{6(13\ ms) + 189\ ms}{300\ ms} = 0.89$$

Por tanto, $f = G[0, 300] \cdot f_0 = 0.89 \cdot 84\ MHz = 74.76\ MHz \rightarrow$ Tomamos la f inmediatamente superior disponible: 75 MHz

El fichero con la implementación de YDS se encuentra en la carpeta del proyecto en: *core/src/main.c*

Cálculo de tiempo de respuesta en el caso peor:

Utilizando la nueva frecuencia de funcionamiento el tiempo de ejecución de cada una de las tareas es el siguiente.

$$C_{locker} = \frac{189\ ms}{75\ MHz} \cdot 84\ MHz = 211.68\ ms$$
$$C_{alarma} = \frac{13\ ms}{75\ Mhz} \cdot 84\ MHz = 14.56\ ms$$

A partir de los tiempos de ejecución podemos comprobar que es planificable.

$$C_{Total} = C_{locker} + 6 \cdot C_{alarma} = 299.04\ ms$$

Justificación de la validez de la verificación:

La implementación YDS no ha incurrido en ninguna modificación de la implementación de las FSMs. Por consiguiente, se garantiza que se mantiene la equivalencia de modelos y la verificación con SPIN sigue vigente.