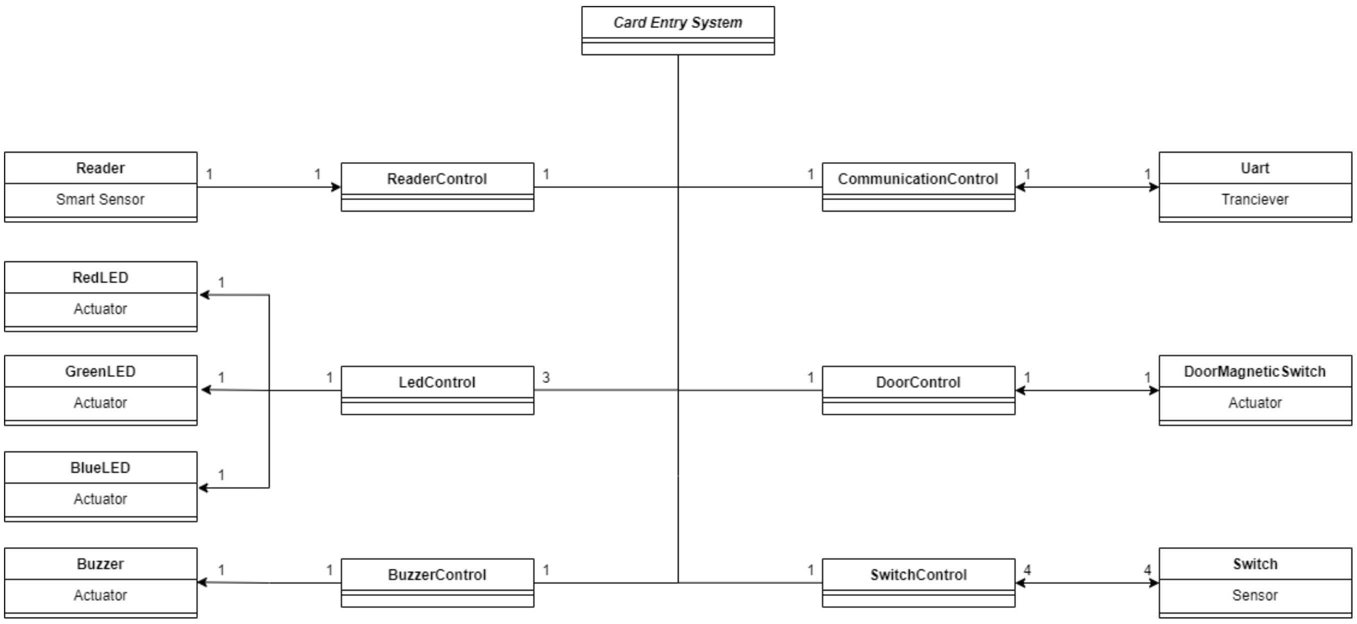
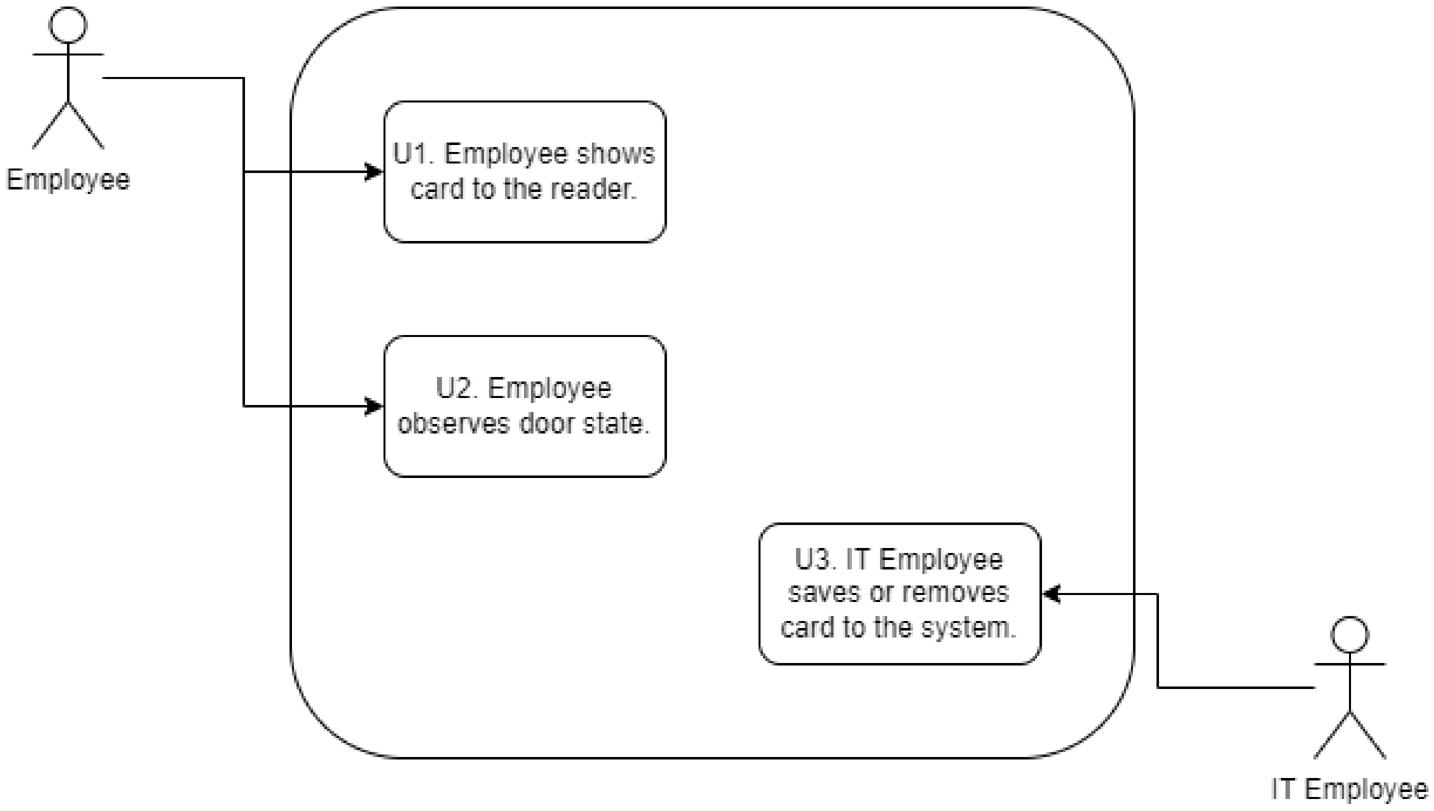
# CARD ENTRY SYSTEM

## 1. Architecture Diagram



**Note**: The system has 4 devices. This diagram shows only one device.

## 2. Use Cases



Employee
U1. Employee shows card to the reader.
U2. Employee observes door state.
U3. IT Employee saves or removes card to the system.
IT Employee

## 3. Scenarios

**Use Case 1:**

- **Scenario 1A:** Employee shows a card the main door’s reader.
- **Scenario 1B:** Employee shows a card the VIP door’s reader.

**Use Case 2:**

- **Scenario 2A:** Employee shows a card when the door is open.

**Use Case 3 :**

- **Scenario 3A:** IT Employee saves a card to the system.
- **Scenario 3B:** IT Employee saves already saved card to the system.
- **Scenario 3C:** IT Employee remove a card from the system.

# 3. Sequence Diagrams

## Use Case 1: Employee shows a card to the reader

### Scenario 1A: Employee shows a card the main door’s reader.

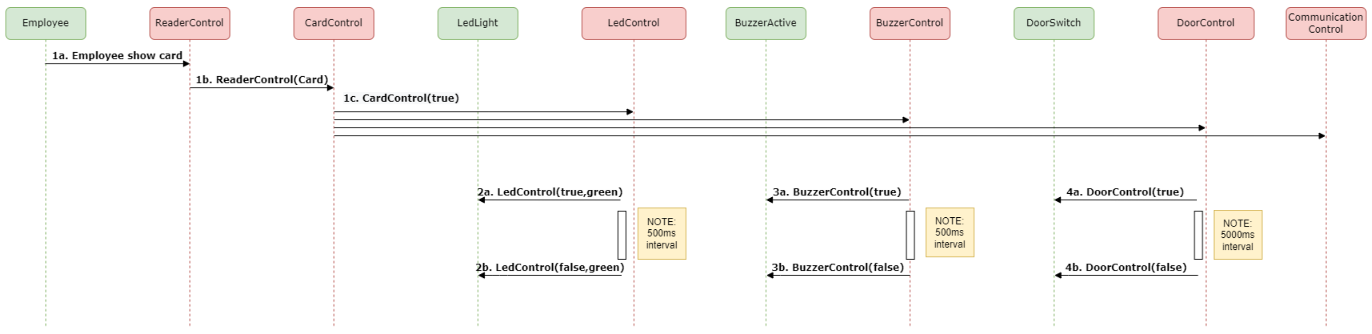
**Pre-conditions:**

- The door is closed.
- The card is already saved.
- Reader is steady state.
- Employee uses her/his card through the main door to enter the building.
- Configswitch state is 0000b

**Scenario :**

1. Employee shows a card to the main door entrance reader.
2. LED blinks green.
3. Buzzer actives during 500ms.
4. The door switch actives during 5s.
5. The card datas send to the system.

**Post-conditions:** The door switch closed when time is up.



**Scenario 1B: Employee shows card the VIP door’s reader.**

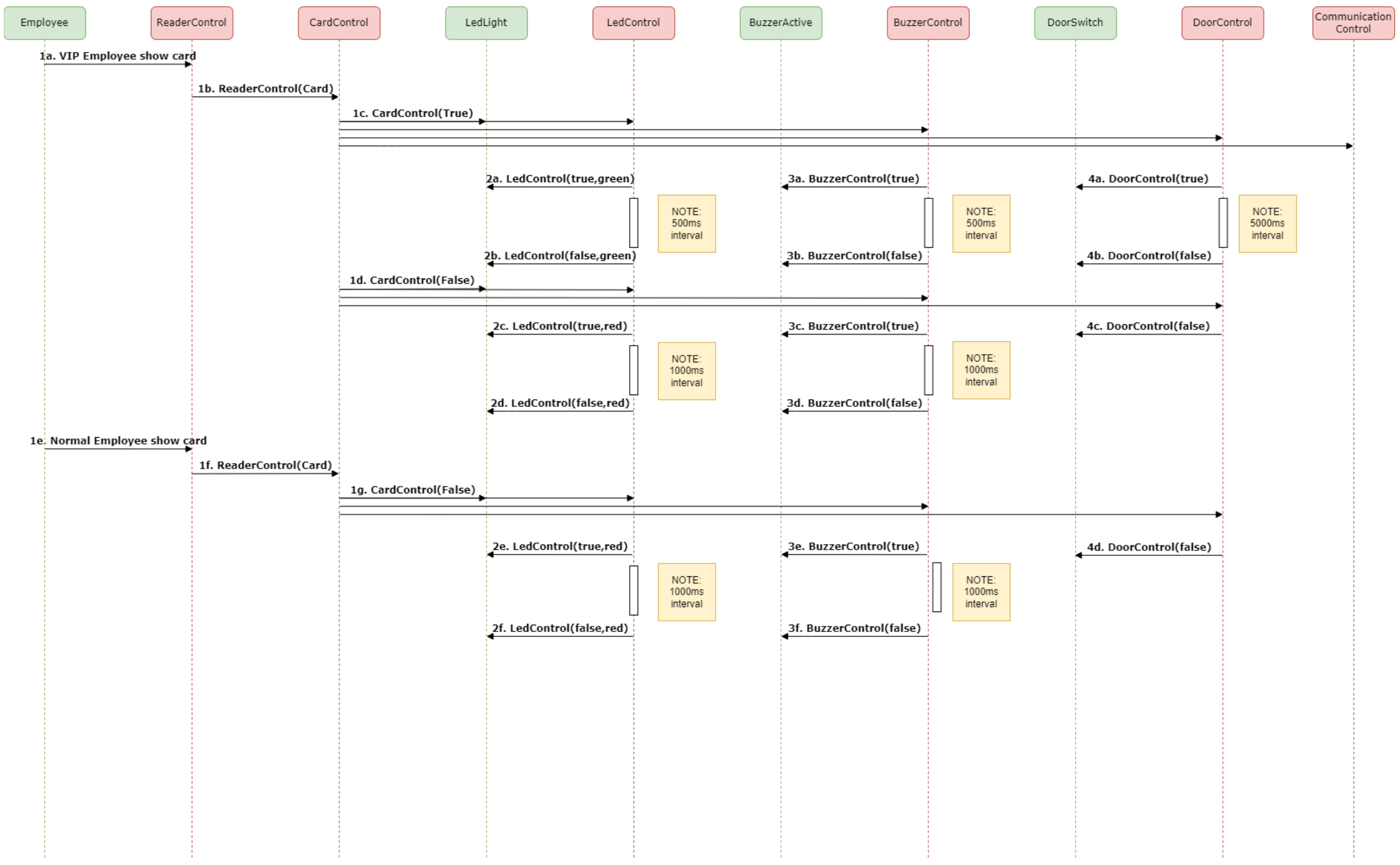
**Pre-conditions:**

- The door is closed.
- The card is already saved.
- Reader is steady state.
- Configswitch state is 0000b

**Scenario :**

1. Employee shows the card to the VIP door’s readers.
2. Depending on the situation, the LED flashes.
3. Depending on the situation, the buzzer actives.
4. Depending on the situation, the door switch actives.
5. The card datas send to the system.

**Post-conditions:** The door switch closed when time is up.



## Use Case 2: Employee observe door's state

### Scenario 2A: Employee shows card when the door is open.

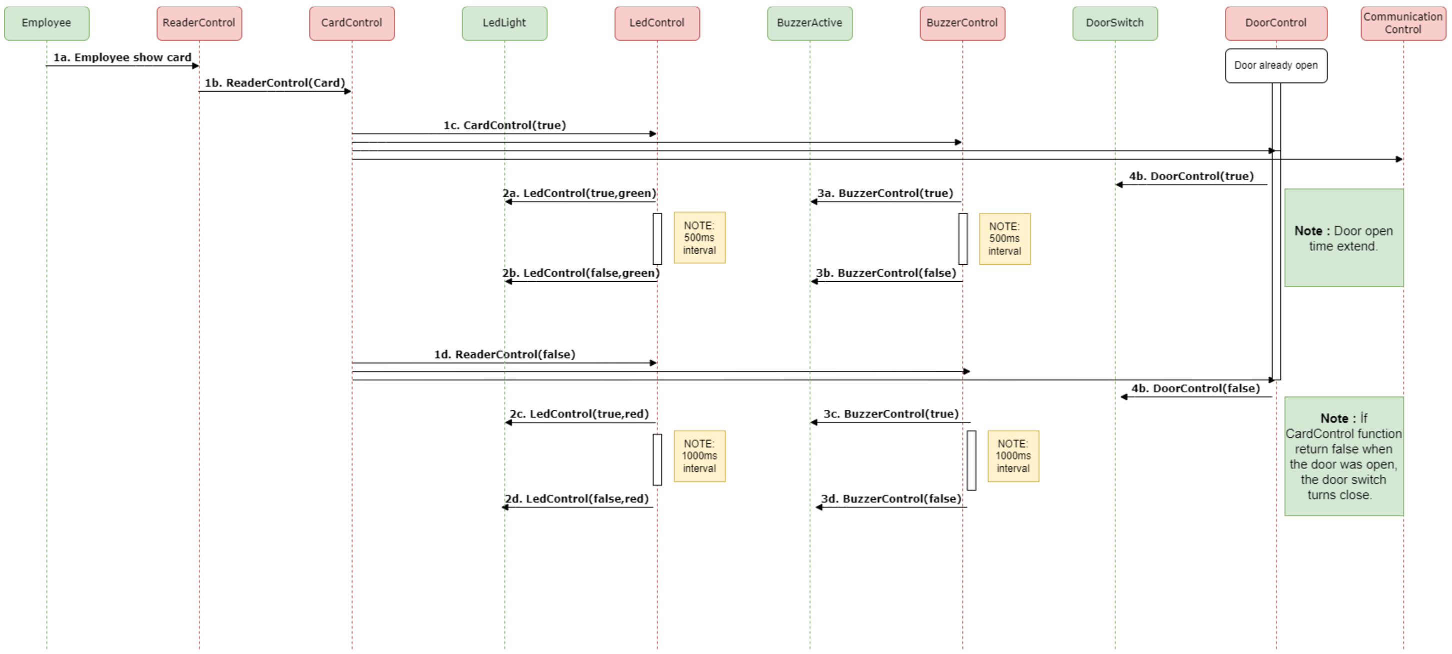
**Pre-conditions:**

- The door is open when the employee uses the card.
- The card is already saved.
- Reader is in steady state.
- Configswitch state is 0000b

**Scenario :**

1. Employee shows the card to the VIP door's reader.
2. Depending on the situation, the LED flashes.
3. Depending on the situation, the buzzer actives.
4. Door switch opens.
5. Card ID sends to the system.

**Post-conditions:** The door switch close when the ReaderControl function returns false.



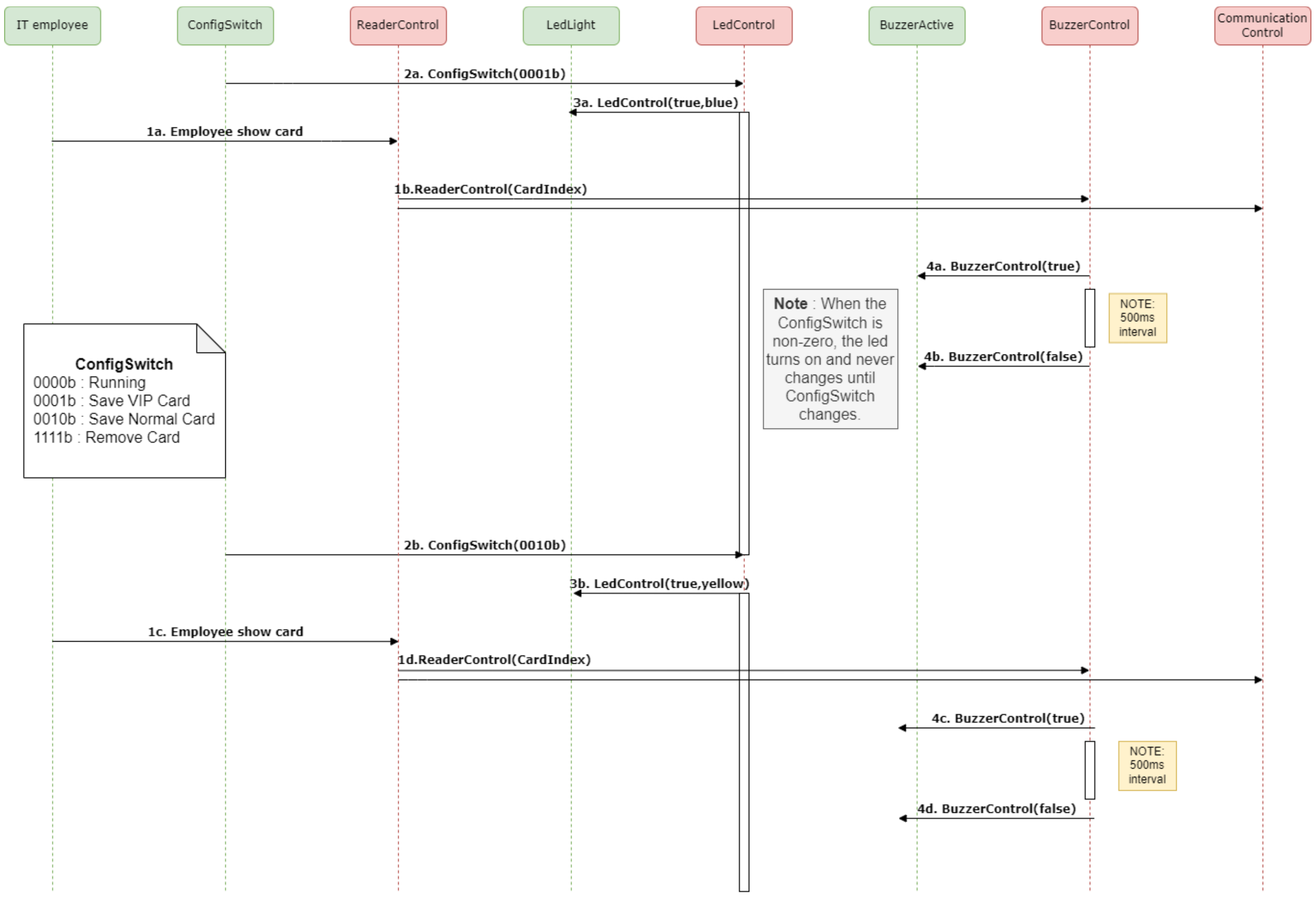
## Use Case 3: IT employee saves card to the system

### Scenario 3A: IT Employee saves card to the system.

**Pre-conditions:**
- ConfigSwitch states are 0001b and 0010b respectively.
- The card is not saved.

**Scenario :**
1. Employee shows the card to any reader.
2. ConfigSwitch value.
3. Depending on the situation, the LED actives.
4. Depending on the situation, the buzzer actives.

**Post-conditions:** The card ID sends to the system and saves from the devices.

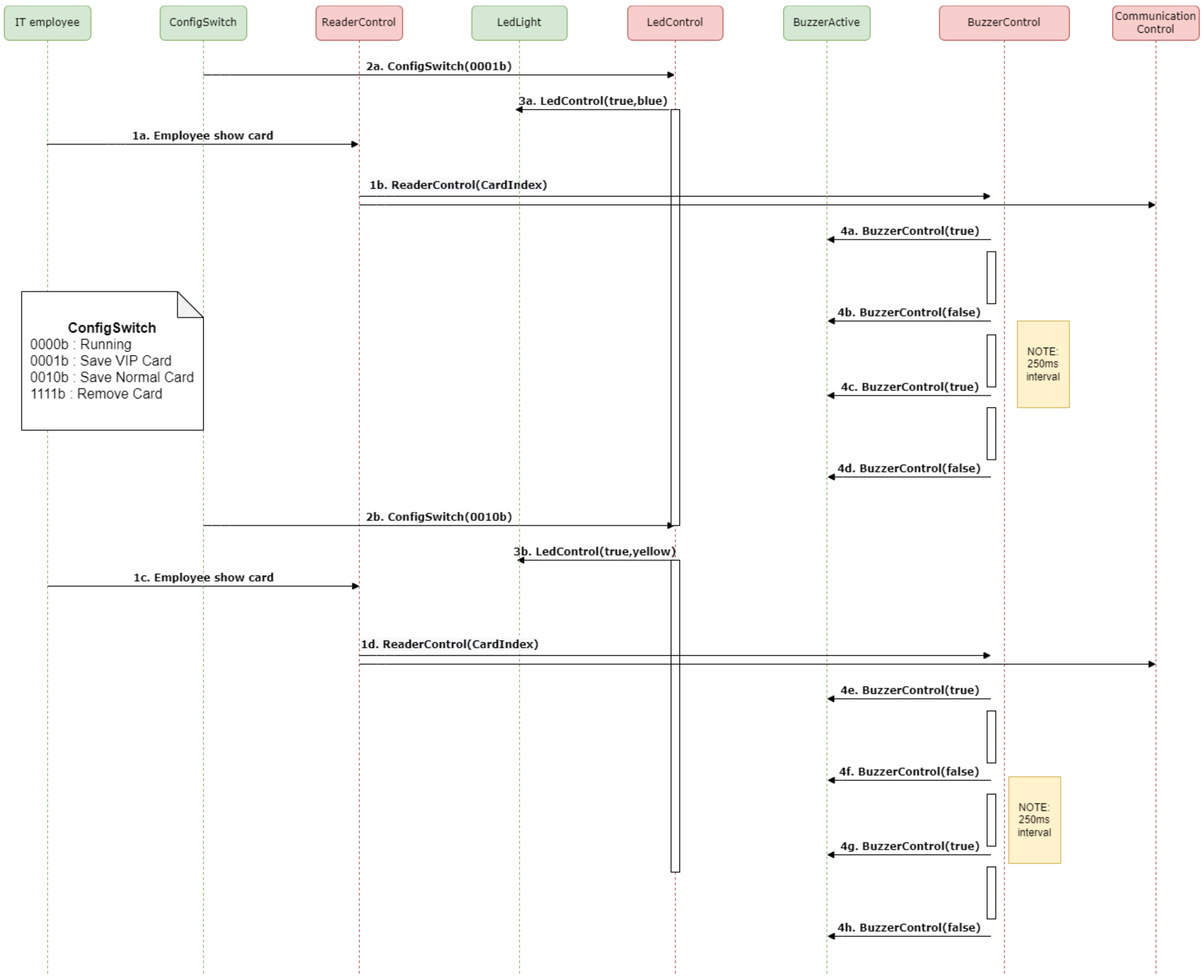


**Scenario 3B: IT Employee saves already saved card to the system.**

**Pre-conditions:**
- ConfigSwitch states are 0001b and 0010b respectively.
- The card is already saved.
- Reader is in config state.

**Scenario :**
1. Employee shows the card to any reader.
2. ConfigSwitch value.
3. Depending on the situation, the LED flashes.
4. Depending on the situation, the buzzer actives.

**Post-conditions:** The card ID sends to the CANBus and saves from the devices.



**Scenario 3C: IT Employee removes the card from the system.**

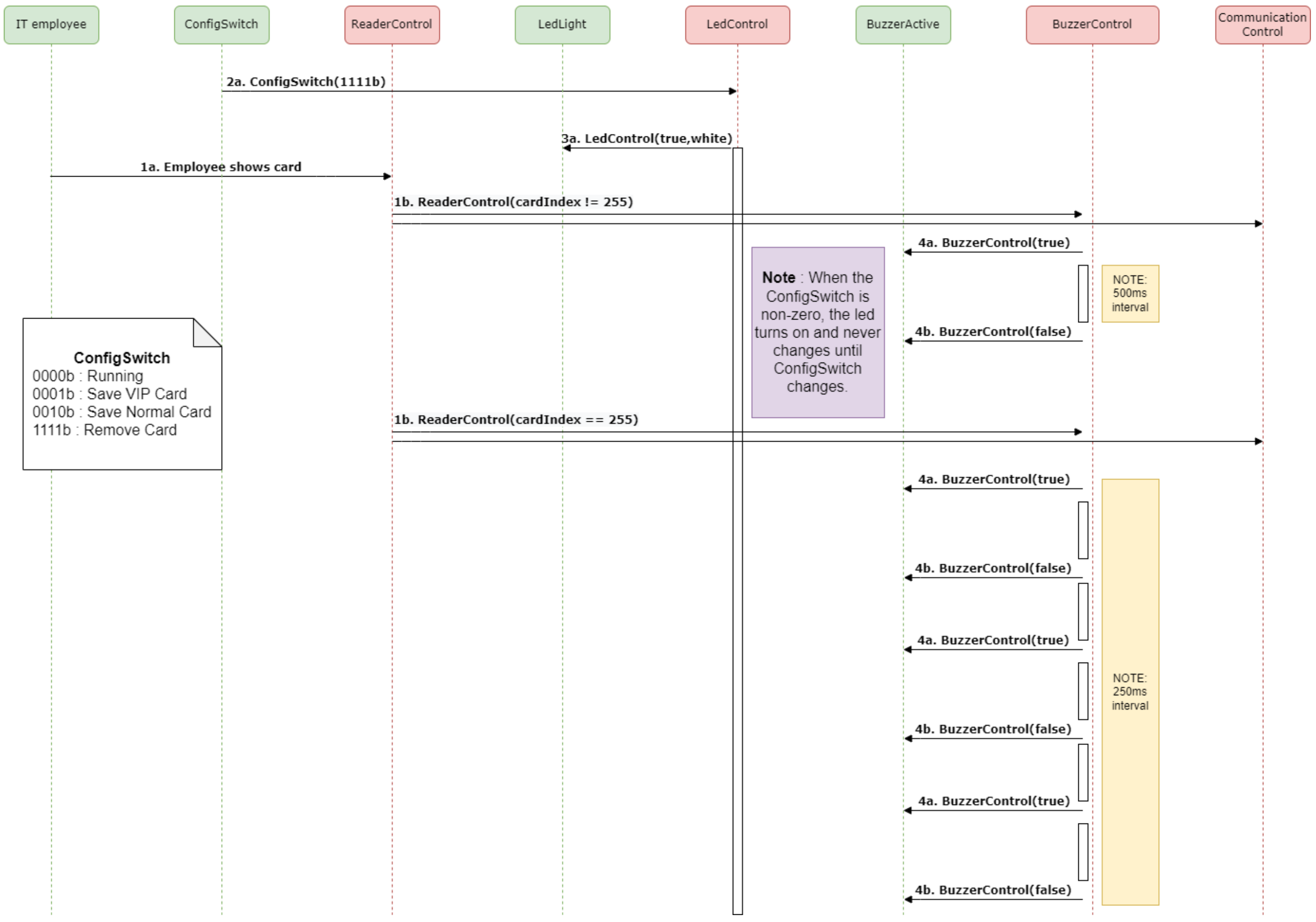
**Pre-conditions:**

- ConfigSwitch state is 1111b.
- The card is saved or unsaved.
- Reader is in config state.

**Scenario :**

1. Employee shows the card to any reader.
2. ConfigSwitch value.
3. Depending on the situation, the LED flashes.
4. Depending on the situation, the buzzer actives.

**Post-conditions:** The removed card ID sends to the CANBus and removes from the devices.



## 4. Behavioral Requirements

### Card  Entry System High-Level Requirements

**R1.** Employee shows the card to the system, the door shall open.
**R2.** The reader reads a card, the led shall turn on depending on the situation.
**R3.** The reader reads a card, the buzzer shall be active depending on the situation.
**R4.** The reader reads a valid card, the system shall send the card ID to the system.
**R5.** Configuration switch state changing shall change the system behavior.

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| Use Case 1 | X | X | X | X | X |
| Use Case 2 | X | X | X | X | 0 |
| Use Case 3 | 0 | X | X | X | X |

**Note: The 0s are expected results.**

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| Scenario 1A | X | X | X | X | 0 |
| Scenario 1B | X | X | X | X | X |
| Scenario 2A | X | X | X | X | 0 |
| Scenario 3A | 0 | X | X | X | X |
| Scenario 3B | 0 | X | X | X | X |
| Scenario 3C | 0 | X | X | X | X |

**Note: The 0s are expected results.**

### Card  Entry System Description

The system consists of several parts. The parts are switch, reader, led and buzzer.

The system can read RFID cards then uses the card IDs to access the building parts.

There are two employee types, Normal and VIP. The normal employee can access only the main door. The VIP employee can access VIP and main doors.

The card saving process is done manually.

### Replication Notation

"[...]" indicates an array of objects or values.

"(...)" indicates a list of values associated with a sensor/actuator. Single-valued inputs/outputs can have the "(" and ")" omitted as a notational convenience.

# System Elements

## Physical State Sensors

These objects are instantiated in the system but do not have a network interface. These objects must be read through the physical interface of a controller.

- **ConfigSwitch(v):** Situation selection switch.
    - v={0x0-0xF}.
    - One 4x1 switch per device.
    - 2 out of 4 switches are used for device ID, the other 2's used for device status.
    - The states corresponding to the switch states are in the table below.

| | |
|---|---|
| 0x00 | Running State |
| 0x01 | VIP Save |
| 0x02 | Normal Card Save |
| 0x03 | Remove Card |
| 0x00 | Outside Door |
| 0x10 | Outside Mid Door |
| 0x20 | VIP Mid Door |
| 0x30 | VIP Door |

## Smart Sensors

These sensor values are available for use by the control system. The below-listed values will correspond to network messages in the implementation phase.

- **ReaderControl(v):** The RFID reader sensor.
    - v={ID}.
    - One RFID sensor per device. The sensor returns the card ID.
    - There are 5-byte ID. 4 out of 5 IDs are pure IDs. The fifth byte is ID's checksum.

## System Actuators

The below-listed values will correspond to network messages in the implementation phase. All actuators are assumed to "remember" their last commanded value and stay there unless commanded otherwise or forced otherwise by system/environment constraints.

- **Led[s](v):** This is a led used to inform the users.
    - v={True, False}.
    - One per type of s corresponding to a color. The colors are red, green and blue. Sometimes the colors use each other to create different colors.
    - All LEDs are set to False at initialization.

- **Buzzer(v):** This is a buzzer used to inform the users.
    - v={True, False}.
    - There is one buzzer per device.
    - Buzzer sets to False at initialization.

## Communication Protocol

UART protocol used in the system.

- **SendData(v):** This protocol used for transmission.
    - v={Tx, Rx}.
    - There is two UART peripheral per device.

## Control System Objects

Environmental objects have already been designed and are provided by the system.

- LedControl[s]
    - One per device (S is an integer [1]..[3])
    - Controls leds
- ReaderControl
    - Checks if the card ID is saved.
- CardControl
    - Checks card information for access to the area.
- BuzzerControl
    - Controls buzzer

## Network Message Dictionary

This section defines the network messages. There is one package for transmit and receive.

### Controller Messages

These messages are sent by the controllers that you will design.  In the later projects, you will be allowed to modify the message dictionary for the controllers in a limited way, but for the time being, you must implement the message dictionary given below:

| Source Node Name | Message Name | Replication | Number of Fields | Description |
|---|---|---|---|---|
| LedControl | mLed | s | 1 | State of led status |
| BuzzerControl | mBuzzer | none | 1 | True when card read. |
| ReaderControl | mReaderID | none | 5 | Card IDs(4 ID + 1 checksum) |
| CardControl | mCard | none | 1 | True when the card has access permission. |
| SwitchControl | mConfig | s | 4 | State of device location and device status. |
| DoorControl | mDoor | none | 1 | True when the card has access permission. |

Uart package size is 17. The package shown below. Some of the messages like mReaderID are in the uart package.

| Package Index | Data | |
|---|---|---|
| 0 | 0xFE | Header |
| 1 | 0xFF | Header |
| 2 | 0xFD | Header |
| 3 | ID1 | Data |
| 4 | ID2 | Data |
| 5 | ID3 | Data |
| 6 | ID4 | Data |
| 7 | Check(ID) | Data |
| 8 | typeOfProcess | Data |
| 9 | CardZone | Data |
| 10 | CardPosition | Data |
| 11 | 0 | Data |
| 12 | 0 | Data |
| 13 | 0 | Data |
| 14 | 0 | Data |
| 15 | PackageID | Data |
| 16 | Check(Package) | Check |

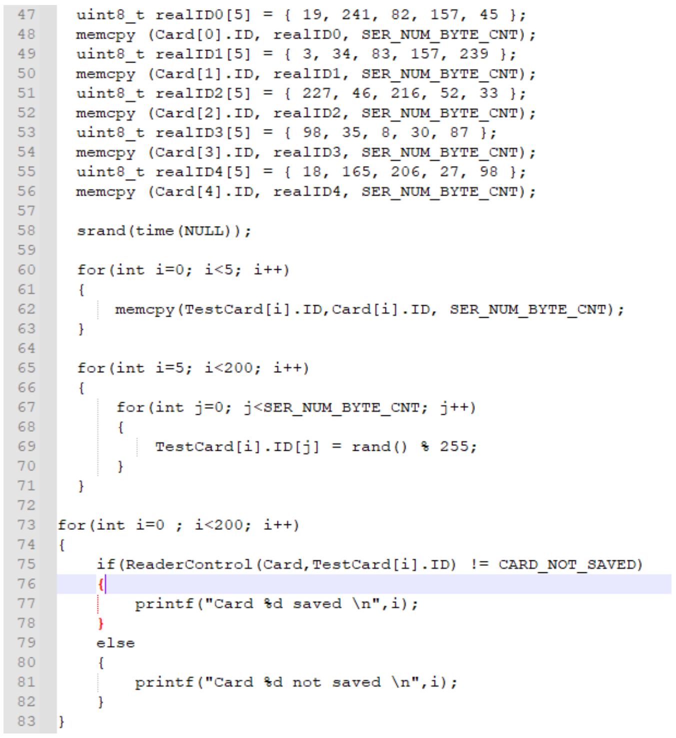
[0 - 3]: Headers
[3 - 7] : Four bytes are IDs and the last byte is the checksum for IDs.
[8] : This byte defines the process type.
[9-10]  : These bytes include card information.
[11-14] : Reserved for future needs.
[15] : PackageID (not used now.).
[16] : All UART package checksum.

# Test

## Unit Test

### "ReaderControl" Unit Test

The ReaderControl function compares the registered card ID with the read card ID. For unit test I created 200 IDs. 5 of them are valid. Rest of them invalid. The verification process was done on the console screen. Test code will be delivered with this document. Test result shown below.



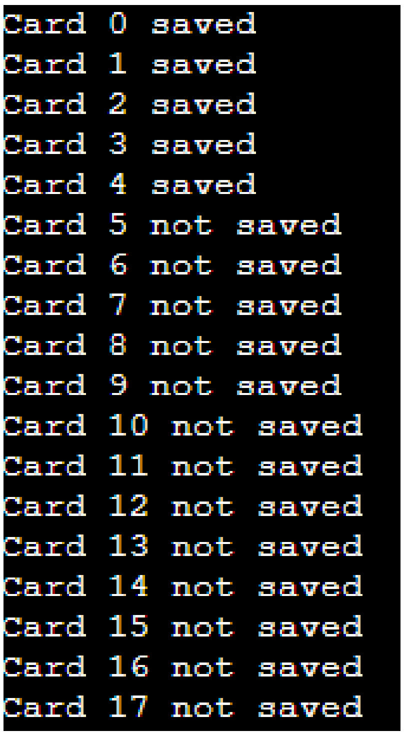
```c
    uint8_t realID0[5] = { 19, 241, 82, 157, 45 };
    memcpy (Card[0].ID, realID0, SER_NUM_BYTE_CNT);
    uint8_t realID1[5] = { 3, 34, 83, 157, 239 };
    memcpy (Card[1].ID, realID1, SER_NUM_BYTE_CNT);
    uint8_t realID2[5] = { 227, 46, 216, 52, 33 };
    memcpy (Card[2].ID, realID2, SER_NUM_BYTE_CNT);
    uint8_t realID3[5] = { 98, 35, 8, 30, 87 };
    memcpy (Card[3].ID, realID3, SER_NUM_BYTE_CNT);
    uint8_t realID4[5] = { 18, 165, 206, 27, 98 };
    memcpy (Card[4].ID, realID4, SER_NUM_BYTE_CNT);

    srand(time(NULL));

    for(int i=0; i<5; i++)
    {
        memcpy(TestCard[i].ID,Card[i].ID, SER_NUM_BYTE_CNT);
    }

    for(int i=5; i<200; i++)
    {
        for(int j=0; j<SER_NUM_BYTE_CNT; j++)
        {
            TestCard[i].ID[j] = rand() % 255;
        }
    }

for(int i=0 ; i<200; i++)
{
    if(ReaderControl(Card,TestCard[i].ID) != CARD_NOT_SAVED)
    {
        printf("Card %d saved \n",i);
    }
    else
    {
        printf("Card %d not saved \n",i);
    }
}
```



```
Card 0 saved
Card 1 saved
Card 2 saved
Card 3 saved
Card 4 saved
Card 5 not saved
Card 6 not saved
Card 7 not saved
Card 8 not saved
Card 9 not saved
Card 10 not saved
Card 11 not saved
Card 12 not saved
Card 13 not saved
Card 14 not saved
Card 15 not saved
Card 16 not saved
Card 17 not saved
```

**"BuzzerControl"** and "**LedControl"** function are basically set an output from the microcontroller. These functions are basically time depending function. In spesific time they  active buzzer or leds. There is not a complex algorithm. These function tests did with debug.

**"CardControl"** function checks if the card has access. There is some information about cards. IDs, status(VIP or Normal), and location. There are three locations: enter, mid and VIP room. This function checks these data and returns true or false. This function unit test did with debugging and used several cards with several IDs, locations, and status. All possible conditions tested.

**"CommunicationControl"** function used for data transmission with UART. This control block test did with debugging too. Sends several data between devices. First, there was no data safety and sometimes some data is lost. For this problem, I added three headers and one checksum.

| **Controller Name** | **Are all the inputs and outputs values checked?** | **Is the boundary values checked?** |
|---|---|---|
| Led Control | yes | yes |
| BuzzerControl | yes | yes |
| ReaderControl | yes | yes |
| CommunicationControl | yes | yes |
| CardControl | yes | yes |

## Integration Test

In the integration test, **“BuzzerControl”**, **“LedControl”**, **“CardControl”**, **”ReaderControl”**, and **“CommunicationControl”** functions tested together in the embedded system.

I had a problem with LEDs and buzzer control. When employee shows his card to the reader over and over, the LEDs and the buzzer active times expand. I set some flags for this problem.

In data transmission, I had to use UART, because my CAN transceiver didn’t work. UART peripheral is a peer-to-peer protocol. I had to use two UART and data transmission was complex.

| Controller Name | Are data transmitted correctly between functions? | Did the functions respond as expected? |
|---|---|---|
| Led Control | yes | yes |
| BuzzerControl | yes | yes |
| ReaderControl | yes | yes |
| CommunicationControl | yes | yes |
| CardControl | yes | yes |