ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA ELECTROMECÁNICA

Itinerario Electrónico

IMPLEMENTATION OF A SMART AUTHENTICATION SYSTEM FOR HOME SECURITY BASED ON NFC AND FACIAL RECOGNITION

Autor: Cristian Gómez Peces

Director: Matthew Smith

Madrid

Julio 2018

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Implementacion of a Smart Authentication System for Home Security based on NFC and Facial Recognition

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2017/2018 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Cristian Gómez Peces Fecha: 25 / 07 / 2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Mathew Smith Fecha: 8 / 2 / 2018

AUTHORIZATION FOR DIGITALIZATION, STORAGE AND DISSEMINATION IN THE NETWORK OF END-OF-DEGREE PROJECTS, MASTER PROJECTS, DISSERTATIONS OR BACHILLERATO REPORTS

1. Declaration of authorship and accreditation thereof.

The author Mr. /Ms. Cristian Gómez Peces

HEREBY DECLARES that he/she owns the intellectual property rights regarding the piece of work: Implementation of a Smart Authentication System for Home Security based on NFC and Facial Recognition that this is an original piece of work, and that he/she holds the status of author, in the sense granted by the Intellectual Property Law.

2. Subject matter and purpose of this assignment.

With the aim of disseminating the aforementioned piece of work as widely as possible using the University's Institutional Repository the author hereby **GRANTS** Comillas Pontifical University, on a royalty-free and non-exclusive basis, for the maximum legal term and with universal scope, the digitization, archiving, reproduction, distribution and public communication rights, including the right to make it electronically available, as described in the Intellectual Property Law. Transformation rights are assigned solely for the purposes described in a) of the following section.

3. Transfer and access terms

Without prejudice to the ownership of the work, which remains with its author, the transfer of rights covered by this license enables:

a) Transform it in order to adapt it to any technology suitable for sharing it online, as well as including metadata to register the piece of work and include "watermarks" or any other security or protection system.

b) Reproduce it in any digital medium in order to be included on an electronic database, including the right to reproduce and store the work on servers for the purposes of guaranteeing its security, maintaining it and preserving its format.

c) Communicate it, by default, by means of an institutional open archive, which has open and cost-free online access.

d) Any other way of access (restricted, embargoed, closed) shall be explicitly requested and requires that good cause be demonstrated.

e) Assign these pieces of work a Creative Commons license by default.

f) Assign these pieces of work a HANDLE (*persistent* URL). by default.

4. Copyright.

The author, as the owner of a piece of work, has the right to:

a) Have his/her name clearly identified by the University as the author

b) Communicate and publish the work in the version assigned and in other subsequent versions using any medium.

c) Request that the work be withdrawn from the repository for just cause.

d) Receive reliable communication of any claims third parties may make in relation to the work and, in particular, any claims relating to its intellectual property rights.

5. Duties of the author.

The author agrees to:

a) Guarantee that the commitment undertaken by means of this official document does not infringe any third party rights, regardless of whether they relate to industrial or intellectual property or any other type.

b) Guarantee that the content of the work does not infringe any third party honor, privacy or image rights.
c) Take responsibility for all claims and liability, including compensation for any damages, which may be brought against the University by third parties who believe that their rights and interests have been infringed by the assignment.
d) Take responsibility in the event that the institutions are found guilty of a rights infringement regarding the work subject to assignment.

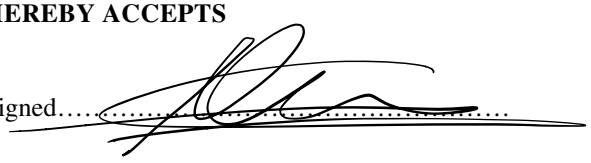
6. Institutional Repository purposes and functioning.

The work shall be made available to the users so that they may use it in a fair and respectful way with regards to the copyright, according to the allowances given in the relevant legislation, and for study or research purposes, or any other legal use. With this aim in mind, the University undertakes the following duties and reserves the following powers:

a) The University shall inform the archive users of the permitted uses; however, it shall not guarantee or take any responsibility for any other subsequent ways the work may be used by users, which are non-compliant with the legislation in force. Any subsequent use, beyond private copying, shall require the source to be cited and authorship to be recognized, as well as the guarantee not to use it to gain commercial profit or carry out any derivative works.
b) The University shall not review the content of the works, which shall at all times fall under the exclusive responsibility of the author and it shall not be obligated to take part in lawsuits on behalf of the author in the event of any infringement of intellectual property rights deriving from storing and archiving the works. The author hereby waives any claim against the University due to any way the users may use the works that is not in keeping with the legislation in force.
c) The University shall adopt the necessary measures to safeguard the work in the future.
d) The University reserves the right to withdraw the work, after notifying the author, in sufficiently justified cases, or in the event of third party claims.

Madrid, on ……25…….. of ……July……..., ……2018……

HEREBY ACCEPTS

Signed……  ……

Reasons for requesting the restricted, closed or embargoed access to the work in the Institution's Repository

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA ELECTROMECÁNICA

Itinerario Electrónico

IMPLEMENTATION OF A SMART AUTHENTICATION SYSTEM FOR HOME SECURITY BASED ON NFC AND FACIAL RECOGNITION

Autor: Cristian Gómez Peces

Director: Matthew Smith

Madrid

Julio 2018

UNIVERSIDAD PONTIFICIA COMILLAS
ESCUELA SUPERIOR TÉCNICA DE INGENIERÍA (ICAI) – INGENIERO INDUSTRIAL

Implementación de un sistema de autenticación inteligente para la seguridad de un hogar basado en tecnología NFC y reconocimiento facial

RESUMEN DE PROYECTO FIN DE GRADO

Autor: Gómez Peces, Cristian

Director: Smith, Matthew

Entidad Colaboradora: University of Michigan

Julio 2018

I. INTRODUCCIÓN

ESTE proyecto se centra en la creación de un sistema embebido controlado por una placa SmartFusion (SoC que incluye FPGA y microcontrolador) a la que se acoplan distintos módulos para garantizar autenticaciones, comunicaciones inalámbricas, seguridad y fiabilidad en el funcionamiento del sistema.

El resumen está dividido en cuatro secciones. En la la sección I se presenta el contexto, el estado del arte y la motivación del proyecto. En la sección II se presentan las funciones y principales características del desarrollo implementado: máquinas de estados, algoritmos de operación, elementos que componen el sistema y funcionamiento en detalle. En la sección III queda reflejado la división de trabajo del proyecto y las tareas realizadas. Finalmente, en la sección IV se reúnen las conclusiones y resultados, y se analiza de forma concisa los objetivos cumplidos tras la integración final de todos los módulos.

El diseño del sistema implementado tiene por objeto facilitar la labor de autenticación a la hora de entrar por una de las puertas de un determinado hogar, manteniendo el mismo nivel de seguridad que con una llave convencional. El sistema evita el uso de un instrumento físico adicional (las llaves), y permite abrir la puerta usando teléfonos móviles con NFC incorporado o mediante la identificación de rasgos faciales de caras usando un módulo de reconocimiento facial. Además de estos dos tipos de autenticación, es necesario implementar un sistema de comunicación que sea seguro (mediante encriptación) tanto por cable como inalámbrico para transmitir credenciales y permisos de acceso.

A. Contexto

La realización del proyecto se divide en dos partes bien diferenciadas, tanto a nivel de desarrollo como de metodología.

La primera parte se realizó durante el primer semestre del curso 2017-2018 en la Universidad de Michigan, como parte del proyecto final del curso *EECS 373: Introduction to Embedded System Design.* La implementación del sistema fue llevada a cabo por los tres miembros que formaron el equipo de desarrollo.

Por otro lado, la segunda parte se realizó como un estudio independiente por cuenta propia y bajo la supervisión del director del proyecto.

En cada parte se desarrollaron distintos módulos que componen el sistema siguiendo distintas metodologías. Mientras que la primera parte se centra en la SmartFusion como principal controlador de acceso, la segunda parte está centrada en la integración del módulo de reconocimiento facial.

B. Estado del Arte

Aunque con algunas características diferentes, el sistema desarrollado reúne las mismas propiedades que los cerrojos inteligentes (también conocidos como smart-locks). Estos cerrojos permiten la apertura de la puerta con la introducción de un código PIN, medidas biométricas (como puede ser huella dactilar) o por NFC [HLMHSW16]. En la Figura 1 se muestra el cerrojo ZKTeco TL400B, un dispositivo que a parte de reunir las características anteriormente mencionadas permite conexión Bluetooth con otros terminales para configurar los términos de acceso.

Figura 1: Cerrojo inteligente de ZKTeco.

El reconocimiento facial aún no está incorporado de forma completa en los sistemas de seguridad que ofrecen las compañías de seguridad o las compañías especializadas en domótica. No obstante, ofrecen servicios y productos relacionados con video vigilancia como pueden ser cámaras IP, utilizadas como elemento visual de acceso remoto del lugar al que apuntan dichos dispositivos. Las conexiones standard usadas para integrar el conjunto del sistema domótico normalmente están basadas en internet (via WiFi o datos móviles) o bluetooth, y son configurables desde una aplicación móvil o web [MR14].

C. *Motivación*

Este proyecto pretende una transición a elementos de seguridad controlados electrónicamente, facilitando el día a día de los usuarios mientras se mantiene los mismos niveles de seguridad en el hogar. En concreto, este sistema esta destinado a hogares con varios propietarios, como bien puede ser una vivienda unifamiliar o en general un hogar de espacios compartidos entre varias personas. Con la instalación de este sistema no sólo se ahorra el uso de llaves físicas, sino que se permite el acceso a propietarios mediante una simple mirada a la cámara de reconocimiento facial, o un simple *swipe* de Smartphone con la funcionalidad NFC activada. Junto con este conjunto de estas funcionalidades que mejoran la sencillez, se encuentra también la posibilidad de habilitar la entrada remotamente a invitados. El poder abrir la puerta fuera de casa a invitados de forma remota es una característica no demasiado extendida y es interesante para personas que entran y salen de casa frecuentemente.

II. DESCRIPCIÓN DEL PROYECTO

El proyecto se puede dividir en cinco entidades que conforman el controlador de acceso: autentificación del identificador NFC, comunicación efectiva entre elementos y controlador, reconocimiento de caras, encriptación de información y medida y mando de los sensores y actuadores. En la Figura 2 se aprecian los distintos módulos integrados.

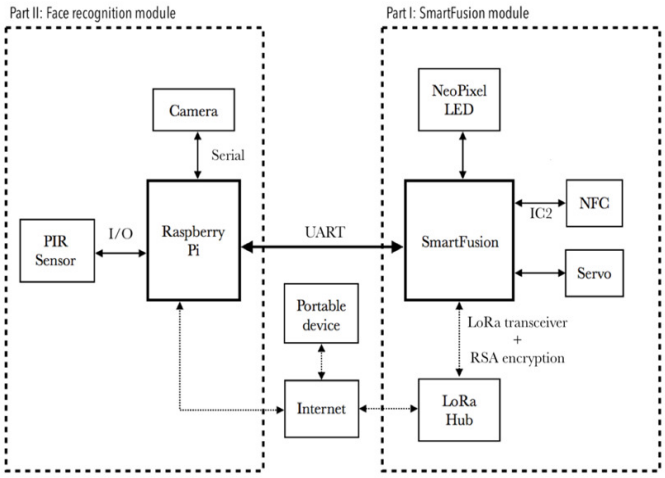


Figura 2: Esquema general de componentes y conexiones.

A. *Elementos que componen el proyecto*

Componente	Nº unidades
Kit de evaluación SmartFusion	1
Transceptor LoRa	1
Hub de LoRa	1
Sensor NFC	1
Interruptor de contacto	1
Servo motor	1
NeoPixel LED	1
Cámara Logitech	1
Raspberry Pi	1
Sensor PIR	1

Tabla 1: Componentes necesarios por puerta

Mientras que todos los componentes del sistema son variables en proporción al número de puertas en las que se instala el dispositivo, sólo se requiere un *hub* de LoRa por casa, usado por las distinta placas SmartFusion para el acceso a internet por medio de los transceptores LoRa. De esta forma se consigue acceso a internet de forma inalámbrica. El servo es el principal actuador del sistema y su principal función es bloquear o desbloquear la puerta. Dependiendo del estado actual, el LED NeoPixel muestra varios colores a modo de *feedback*. El sensor NFC y la cámara se usan como elementos de autentificación por campos electromagnéticos y reconocimiento facial, respectivamente.

El sensor PIR es utilizado como detector de personas para que la Raspberry Pi pueda comenzar con la ejecución del algoritmo de reconomiento de caras.

B. *Funcionamiento*

El controlador principal de acceso se muestra en la Figura 3. Este control incluye la identificación por NFC, encriptación de datos usando encriptación RSA

implementada en la FPGA, conexión al servidor online con las credenciales requeridas para permitir el acceso, control del servo y del NeoPixel LED.

Tras la inicialización de los componentes, el sistema comprueba el estado de la puerta. Si está abierta y ha finalizado el tiempo máximo de apertura, la puerta se cierra automáticamente cambiando el color del LED de amarillo a verde. En caso de que este tiempo no haya pasado, se mantiene simplemente el color amarillo del LED. Por otro lado, si la puerta está cerrada, se realizan dos comprobaciones. En primer lugar, se realiza identificación de radio frecuencia usando el sensor NFC. Si se detecta un Smartphone o una *NFC tag* se procede con la solicitud de credenciales al servidor web montado en el *hub* de LoRa a través de una conexión a internet por medio de los transmisores LoRa desde la SmartFusion.

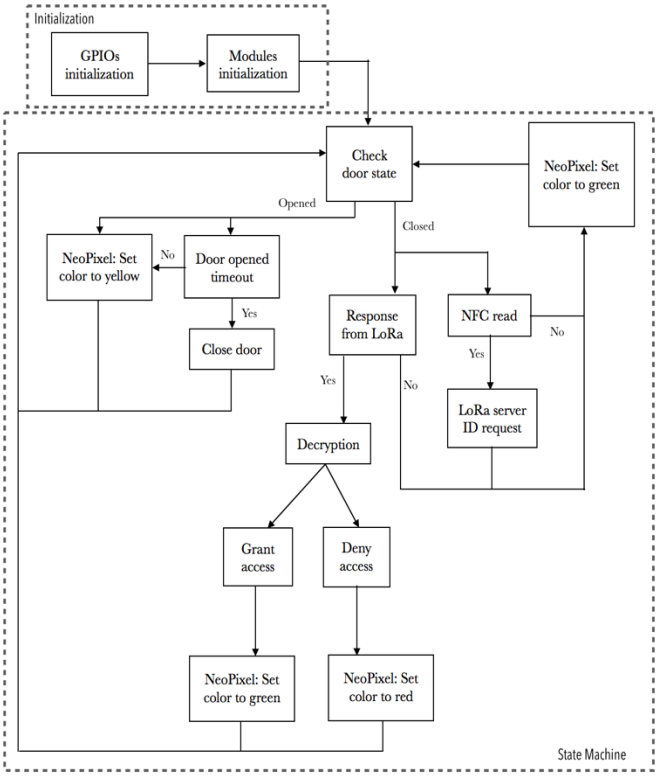


Figura 3: Esquema de funcionamiento.

En segundo lugar, se comprueba si se ha recibido respuesta desde el servidor con la aceptación de credenciales (en caso de que el usuario esté en la lista blanca de identificadores o se haya concedido acceso remoto) o con denegación (en cuyo caso, bloquea la puerta para no permitir el acceso). Como se puede deducir, hay tres posibilidades una vez se ha leído la identificación de NFC: ID perteneciente a la lista blanca de identificadores, perteneciente a la lista negra, o ninguna de las dos. En el primer y segundo caso de abre o se cierra la puerta respectivamente. En el tercer caso, se espera a que alguno de los usuarios de la lista blanca conceda acceso o lo deniegue. En caso de desbloqueo de la puerta, el color del LED se cambia a amarillo. Si se bloquea, pasa a color rojo durante unos segundos.

Por otro lado, se encuentra el módulo de reconocimiento facial. Esta operación es realizada por la Raspberry Pi usando OpenCV como librería esencial de *Computer Vision* para poder ejecutar el algoritmo de reconocimiento. Toda la implementación llevada a cabo en el lado de la Raspberry Pi está realizada en Python, mientras que en la SmartFusion por un lado el software está escrito en C y el hardware en Verilog.

El flujo de operación de este módulo viene detallado en la Figura 4.

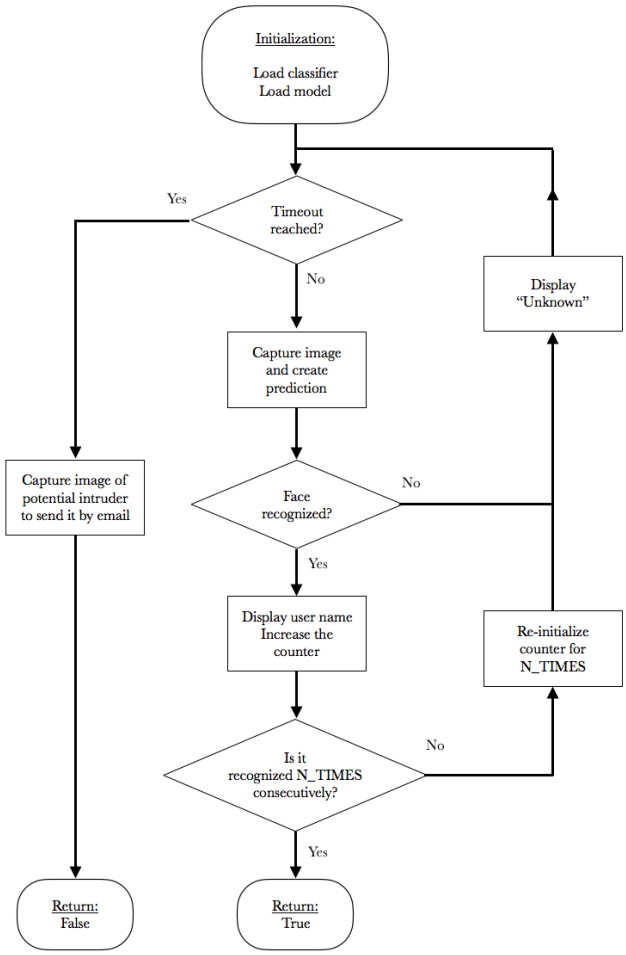


Figura 4: Algoritmo de operación de reconocimiento

La inicialización se ejecuta cargando el clasificador y el modelo que ha sido previamente elaborados. El algoritmo de reconocimiento usado para este proyecto es LBP (*Local Binary Patterns*), por ser uno de los más extendidos y mejor documentados [AHP06]. Cada cierto

tiempo se realiza una lectura en el sensor PIR para determinar si alguien está enfrente de la cámara. En dicho caso, se realiza consecutivamente una serie de fotografías a la persona en cuestión y, si se encuentra en el modelo precargado, se muestra el nombre que le identifica. Si el sujeto es reconocido N veces en las sucesivas capturas, se determina que la cara pertenece a algún usuario de la casa. En caso contrario, se procesa como desconocido y tras un tiempo determinado de ejecución, si el algoritmo sigue sin reconocer al sujeto se toma una captura de la persona y es enviada por email a la lista de propietarios del hogar. En cualquier caso, la Raspberry envía a la SmartFusion la información del resultado del reconocimiento de forma encriptada usando XTEA como algoritmo de encriptación y UART como protocolo serie de comunicación.

III. METODOLOGÍA

La primera parte del proyecto se realizó en equipo, y es por ello por lo que se hizo necesario usar una herramienta de control de versiones para implementar los desarrollos de forma independiente y escalada. La herramienta elegida fue *GitHub*, por familiaridad con su uso. A medida que se implementaban cambios y se realizaban pruebas, éstas quedaban reflejadas en los *commits* cada 2 o 3 días.

En la segunda parte, dado que se trató de un trabajo por cuenta propia, se disponía de mayor libertad. Las implementaciones en Python se fueron realizando en primer lugar en el ordenador personal. Finalmente, tras realizar varias pruebas locales se procedía con su correspondiente implementación en la placa.

A. *Objetivos*

Los objetivos concretos del proyecto son:

- Conseguir una efectiva comunicación entre el controlador de acceso y distintos dispositivos conectados. Esto es, crear todas las librerías necesarias para controlar los módulos de NFC, LoRa y demás dispositivos.
- Hacer del proyecto un sistema protegido frente a ciberataques. Es decir, encriptar toda comunicación y traspaso de datos que sea vulnerable.
- Correcta integración: conseguir un sistema basado en una máquina de estados que sirva de forma efectiva para bloquear/desbloquear la puerta según las especificaciones anteriormente expuestas. Esto es, el sistema debe poseer las capacidades (entre otras) de leer NFC tags, comunicarse de forma inalámbrica, cambiar el color de estado, reconocimiento facial y tener todo ello ejecutándose sin errores o con una mínima probabilidad de fallo.
- Desarrollar y aplicar un algoritmo fiable de reconocimiento facial.

B. *División del Trabajo*

El desarrollo de la primera parte se dividió entre los tres miembros del equipo. En concreto, el autor se encargó de la implementación de los drivers y librerías a bajo nivel del sensor NFC, tests de conexión entre transmisores y *hub* de LoRa, soldadura de aquellos componentes en los que se requería, de la inicialización de hub de LoRa y de los planos de diseño en SolidWorks de la puerta y marco de la puerta destinados a imprimirlos para la exposición final. Adicionalmente, los miembros del grupo trabajaron conjuntamente en la integración final y realización de pruebas.

Toda la implementación realizada durante la segunda parte se realizó por cuenta propia, guiada por el director de proyecto en los hitos realizados a lo largo del semestre.

C. *Tareas realizadas*

A continuación, se muestran las tareas realizadas durante los dos periodos de desarrollo.

Primera parte:

- Decisión de los componentes a usar: *brainstorming* de los componentes necesarios para poner en practica la idea del proyecto.
- Compra y adquisición de los diferentes elementos que componen el sistema.
- Elaboración de librerías para el control del sensor NFC.
- Inicialización del *hub* LoRa.
- Lograr la comunicación entre SmartFusion y transmisores LoRa.
- Lograr comunicación entre transmisores LoRa y el *hub* de LoRa.
- Implementación de la encriptación RSA en la FPGA.
- Diseño e implementación de una web en el servidor para controlar y permitir accesos remotos.
- Diseño de puerta y marco de puerta para su impresión 3D en la exposición final.
- Integración de módulos y pruebas
- Documentación del proyecto
- Exposición final el 14 de diciembre de 2017.

Segunda parte:

- Investigación sobre las herramientas a usar para realizar reconocimiento facial.
- Implementación de las tres fases de reconocimiento facial.
- Pruebas de efectividad en el reconocimiento

cara de múltiples sujetos.

- Adaptación de algoritmos a Raspberry Pi, con cambios sujetos a las diferencias entre versiones de OpenCV.
- Desarrollo de *timer* para el control del flujo de operación en la ejecución de programa principal y subprogramas.
- Implementación de módulo memoria para la administración local de datos del sujeto a identificar.
- Desarrollo de librerías (para ambas placas) para la comunicación por UART con la SmartFusion en la que se incluye encriptación usando XTEA.
- Realización de pruebas de la encriptación en Python y en C.
- Integración final de todos los módulos.
- Demostración final y verificación de funcionamiento correcto.

IV. RESULTADOS Y CONCLUSIONES

El prototipo utilizado para la realización de pruebas finales se muestra en la Figura 5. En la ilustración aparece la placa SmartFusion a la izquierda y el transmisor LoRa a la derecha. Este sistema cumple los objetivos principales con los que se inició el desarrollo de este proyecto, aunque unas metas fueron alcanzadas de una forma más satisfactoria que otras.

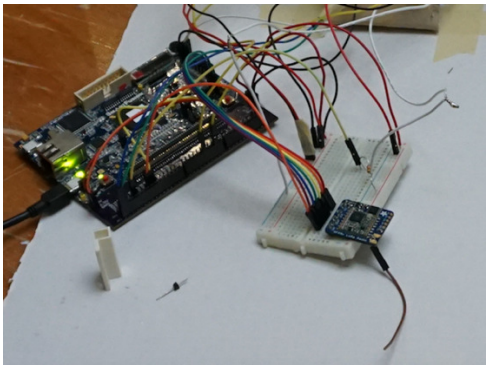


Figura 5: Prototipo final.

La interconexión con cada uno de los componentes fue un éxito. La creación de las librerías permite acceder y tener un mayor control sobre el dispositivo a bajo nivel. El desarrollo implementado para el manejo de las funcionalidades de cada elemento funciona de manera eficaz, robusta y flexible, dada la ausencia de fallos en las pruebas finales y a la implementación propia de drivers.

Los algoritmos de encriptación implementados (RSA y XTEA) son capaces de realizar sus funciones de manera adecuada. No obstante, se esperaba poder haber podido implementar una encriptación RSA de forma más segura, añadiendo algún protocolo de comunicación adicional o con mayor número de bytes, al menos 256 (la versión implementada es sólo de 64 debido a limitaciones en el número de módulos disponibles en la FPGA). No obstante, sirve igualmente de prueba de concepto.

La integración final se realizó de manera exitosa. La placa SmartFusion es capaz de lidiar con todos los procesos, interrupciones y algoritmos perfectamente. Todos los elementos del proyecto funcionan de acorde a lo esperado de manera eficaz, aunque en ocasiones aisladas con ligeros retrasos de respuesta.

El algoritmo de reconocimiento facial funciona perfectamente. Los modelos predictivos funcionan mejor cuantos mas datos (imágenes variadas de la cara del sujeto) se tengan. No obstante, este módulo se puede implantar de forma efectiva ajustando el nivel de correlación (entre modelo y sujeto de prueba) adecuadamente.

Por otra parte, aunque todos los objetivos se han conseguido, quedan varios elementos del proyecto por mejorar. El diseño del sistema está pensado para trabajar en condiciones ideales. Sin embargo, cualquier intruso potencial podría desconectar la corriente o proporcionar una fuerte descarga sobre el cerrojo para inutilizarlo. Esto es, habría que incluir medidas de protección eléctricas. Otro de los aspectos olvidados durante la ejecución de este trabajo es la experiencia de usuario. La interfaz gráfica de aplicación web del servidor es muy simple como para ofrecerse a cliente. Además, el sistema de aviso desde la Rasperry Pi mediante e-mail es independiente del servidor. Sería conveniente fusionar ambas funcionalidades y crear una aplicación móvil desde la que poder gestionar toda la configuración de manera sencilla.

En conclusión, aunque hay determinados aspectos a mejorar, todos los objetivos planteados al comienzo del proyecto han sido logrados satisfactoriamente, dando lugar a un robusto, eficaz e innovador sistema de autentificación por credenciales.

REFERENCIAS

[MR14] T. Miori and D. Russo. "Domotic Evolution towards the IoT". In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, Victoria, BC, 2014, pp. 809-814. doi: 10.1109/WAINA.2014.128

[HLMHSW16] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song and D. Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*. ACM, 2016. pp. 461-472.

[AHP06] T. Ahonen, A. Hadid and M. Pietikainen. "Face Description with Local Binary Patterns: Application to Face Recognition". In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037-2041, Dec. 2006. doi: 10.1109/TPAMI.2006.244

UNIVERSIDAD PONTIFICIA COMILLAS
ESCUELA SUPERIOR TÉCNICA DE INGENIERÍA (ICAI) – INGENIERO INDUSTRIAL

Implementation of a smart authentication system for home security based on NFC and facial recognition

PROJECT SUMMARY

Author: Gómez Peces, Cristian

Director: Smith, Matthew

Collaborating Entity: University of Michigan

July 2018

I. INTRODUCTION

THE work presented hereby discusses the viability of creating a microprocessor-based embedded system controlled by a SmartFusion board (SoC that includes both the FPGA and microcontroller). Within the system, multiple modules are fitted together aiming to secure authentications, wireless communications, security, and system functioning reliability.

This summary is divided into four sections. The project context, state of the art, and motivation are presented in section I. Section II presents the functions and main characteristics of the project development: state machines, operation algorithms, elements that make the system up and functioning in detail. Section III presents the division of labor as well as the tasks performed along the different stages of the project. Section IV concludes with results and conclusions obtained from the final integration. In this section all the goals accomplished are briefly analyzed as well as additional approaches to improve the system.

The design of the embedded system aims at improving the authentication when entering the home and keeping the entry security as safe as it usually is with an ordinary key. The system avoids the use of a specific instrument to open the door (physical keys) but may open the door by using smartphones with NFC capabilities activated or with the user main face characteristics identification. In addition to these two types of authentication, the data security while communicating between modules should be secured, with the use of different types of encryption both for wired and wireless communications in order the send and receive credentials and access permissions.

A. Context

The project is divided into two parts perfectly distinguishable, both in development and methodology terms.

The first part was carried out during the first term of the University of Michigan 2017-2018 academic year, as a final project for *EECS 373: Introduction to Embedded System Design*. The whole implementation was done by the three members of the development team.

The second part was carried out as an independent study and under the supervision of the director of this project.

Each part presents the implementation of the multiple modules that compose the system following particular methodologies. While the first part is focused on the SmartFusion development as the main access controller, the second part is oriented to the integration of the facial recognition module.

B. State of the Art

Even though there are slight differences, the system that was developed gathers most of the features that the smart locks normally have. These locks are normally incorporated into the door bolts and they can unlock the door using PIN codes, biometrics measures (like fingerprints) or with NFC IDs [HLMHSW16]. The data provided should match with the expected. Figure 1 shows a ZKTeco TL400B smart-lock, a device that not only has all the features previously mentioned, but also has Bluetooth connection with other terminals in order to configure access terms.

Figure 1: ZKTeco Smartlock.

Face recognition is not yet fully incorporated in forced entries prevention systems that usually security and domotics specialized companies offer. However, other services are provided such as IP cameras, which are used as remotely accessible devices to provide real time visual checks. Today, most of standard connections made with domotics purposes are based on wireless communications through Wi-Fi, internet data, or Bluetooth. These let us configure any device from a smartphone app or web [MR14].

C. *Motivation*

The aim of this project is the acceleration of the transition to security elements electronically controlled, making life much easier while maintaining the same levels of home security. This system is particularly oriented to shared houses/apartments, in which there are multiple owners living together. With the installation of the developed system, not only is the use of keys avoided, but it allows access with a mere look to the facial recognition camera or a simple smartphone swipe with the NFC functionality activated. In addition to this set of features that improves the simplicity of entries, we find the possibility of unlocking the door remotely for guests. This feature is not commonly used and is interesting for people who may get in and out frequently.

II. PROJECT DESCRIPTION

The whole implementation can be divided into five main parts: authentication of the NFC ID, effective communication between modules and microcontroller, faces recognition, encryption of the information transmitted between devices, and measures and control signals from sensors and actuators respectively. Figure 2 shows a diagram with the connections between modules and the protocols used to interface them. The Raspberry Pi communicates with the SmartFusion through UART using a XTEA encrypted channel. I^2C is the protocol chosen to interface the NFC sensor and the communication with the LoRa transmitter is driven using SPI protocol.

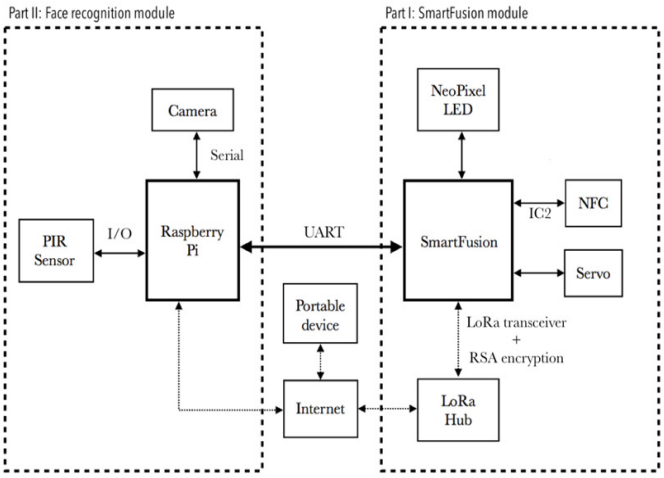


Figure 2: Diagram of access controller connections.

A. *Elementos que componen el proyecto*

Component	Number of units
SmartFusion Evaluation Kit	1
LoRa transceiver	1
LoRa gateway	1
NFC sensor	1
Contact switch	1
Servo	1
NeoPixel LED	1
Camera Logitech	1
Raspberry Pi	1
PIR sensor	1

Table 1: Components required per door

While the number of the components required is proportional to the number of doors in the house, only one Lora hub is needed in the whole house, since it can be used by multiple SmartFusion boards to get Internet access through the LoRa transceivers. This is basically the approach used to connect to the internet wirelessly after all the data to be transferred is encrypted (using RSA encryption). The servo motor is the main actuator of the system and its function basically consists of locking or unlocking the door. Depending on the current state of the door, the NeoPixel LED will show different colors in order to provide visual feedback. The NFC sensor and the camera are the devices in charge of providing identifications to authenticate the person as a house owner by using RFID and facial recognition, respectively. The PIR sensor is used to let the Raspberry know whether or not there is a person in front of the door, so the face recognition algorithm execution can begin without leading to losses of energy due sampling.

B. Operation

The main controller operation of the door access is shown in Figure 3. This control includes NFC identification, data encryption using RSA Montgomery's algorithm which is implemented in the FPGA, connection to the online server with the required credentials, servo control and communication with the NeoPixel LED.

Once the modules are initialized, the system verifies the door state. If it is opened and it has reached a certain timeout, the controller automatically closes the door and changes the color of the LED from yellow to green. In case the timeout is not reached, the color of the NeoPixel LED remains yellow. On the other hand, if the door is closed, the SmartFusion proceeds with two different checks. Firstly, a radio frequency identification is carried out by the NFC sensor. If it detects a smartphone or a NFC tag, the controller requests credentials to the web server that are set on the LoRa hub.

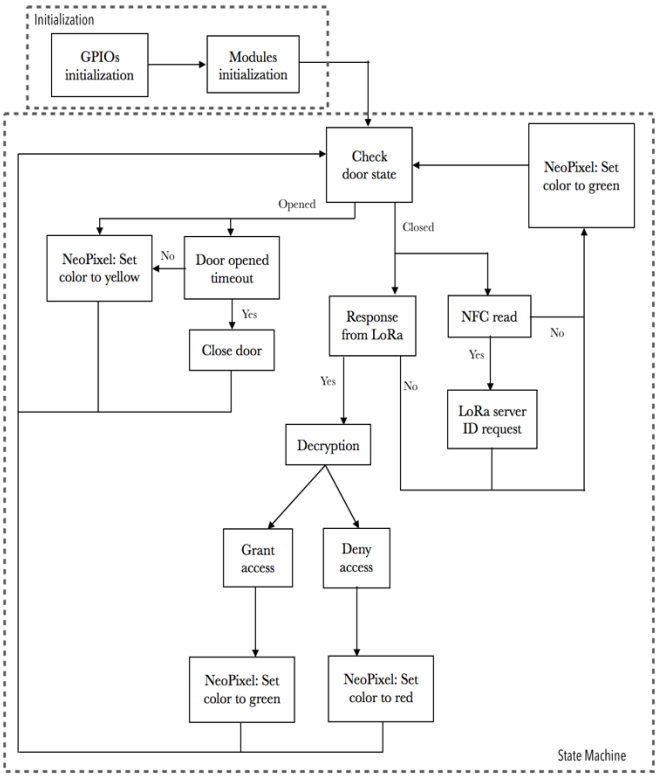


Figure 3: Main access controller operation diagram.

Secondly, the Cortex-M3 verifies if any response was received from the servers, grating or denying the access to the access requester. The request can be rejected (in case the user ID is on the blacklist) or the controller can grant the access, as long as the user ID is on the whitelist. As can be deduced, there are three different possibilities once the ID is read: the ID belongs to the whitelist, it belongs to the blacklist or it belongs to neither. In the first and second case, the response is direct: the microcontroller will outright grant or deny the access. In case the ID does not belong to any of the list, the system waits until a whitelist user lets the guest into the house, by granting the access remotely through the web server application. In case the door is unlocked, the color of the LED changes to yellow. In case the door is blocked (due to a blacklist user), the color changes to red for several seconds.

Moreover, the face recognition module is performed by the Raspberry Pi module. This operation is carried out using the open source library OpenCV, which includes essential Computer Vision classifiers and methods for executing the recognizing algorithm. All the development in the Raspberry Pi was done in Python, unlike the SmartFusion which was done in C (for the microcontroller) and in Verilog (for the FPGA).

Figure 4 shows a detailed diagram of the workflow operation of the face recognition algorithm.

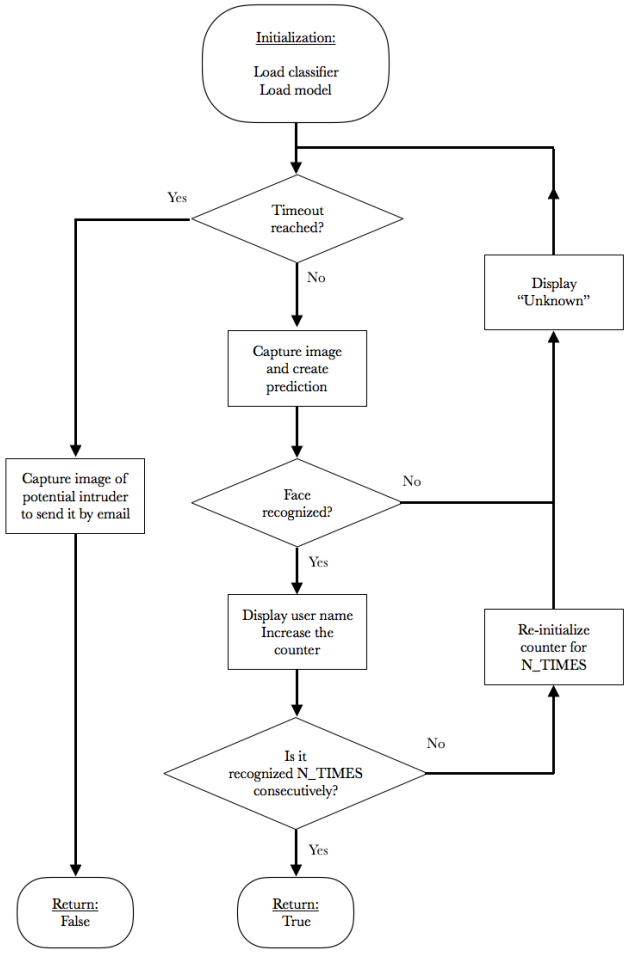


Figure 4: Workflow operation diagram.

The initialization starts executing the frontal face

classifier and the previously made face model. The algorithm used in this project for the face recognition task is LBP (*Local Binary Patterns*), since this is one of the most extended and documented algorithms available [AHP06]. After a certain timeout, the Raspberry performs a sample using the PIR sensor to detect whether or not there is a person in front of the door. If so, multiple pictures are consecutively taken and if face characteristics match with the one inside the preloaded model the name of the person is shown on screen. If this subject is recognized after N consecutive times, he/she is processed as a whitelist user. Otherwise, the system processes the face as unknown, takes a picture of the potential intruder, and sends it by email to all the owners of the place. In case a person is detected, the Raspberry Pi sends the information of the recognition result over a cable using XTEA as encryption method and UART as serial communication protocol.

III. METHODOLOGY

The first part of the project was done in team, and therefore it was necessary to use a version control tool in order to submit each one's development independently. The tool chosen for version control was GitHub, because the members of the team were familiar with it. Every implementation was committed and pushed after testing it, approximately every 2 or 3 days.

During the second part, since it was an independent study, there was more freedom in the methodology of working. The implementations in Python were carried out first on the personal computer. Finally, after performing some local tests this software was implemented on the board.

A. Objectives

The specific objectives of the project are:

- Interface all devices and sensors correctly. This means, the board should be communicating with the components with low rates of failure. The system should be aware continuously of what is going on in real time.
- Secure the system. Build a system that is protected against cyber-attacks. All communications and data transfers that may be vulnerable should be encrypted.
- Effective integration. The final integration should be based on a state machine that is capable of locking and unlocking the door according to the specifications previously mentioned. In other words, the system should be able to perform task such as reading NFC tags, wireless communications, changes of LED color based on door state, or face recognition (among others) with a minimal chance of error.
- Develop an effective and reliable face recognition algorithm.

B. Division of labor

The work carried out during the first part was divided into specific tasks to be done by each team member. In particular, the author was in charge of the drivers and low level libraries of the NFC sensor, connection tests between LoRa transmitters and hub, LoRa gateway initialization, and the design of the 3D printed door and door frame for the final exposition with SolidWorks software. In addition, the team members worked together in the final integration, testing, and poster printing.

All the development carried out during the second part was carried as an independent study under the guidance of the director of the project at several milestones that were set along the winter semester.

C. Completed tasks

The tasks done during the whole period of development are shown below

First part:

- Decision-making of the components to use: *brainstorming* of the required components to start the idea of the project.
- Purchase and acquisition of all the components that compound the system.
- Library development to control the NFC sensor.
- LoRa hub initialization.
- Achieve communication between SmartFusion and LoRa transceivers.
- RSA encryption implementation and optimization on the FPGA.
- Design and deployment of web app on the server in order to control and grant access remotely.
- Design of door and door frame for its 3D printing for the final exposition.
- Modules integration
- Project documentation
- Final exposition on 14th December 2017.

Second part:

- Research to find the most appropriate tools to perform facial recognition.
- Development of the three phases of the face recognition algorithm.
- Testing of the facial recognition with multiple subjects.

- Adjustments of the algorithm when adapting it to the Raspberry Pi. These changes were necessary because of software changes in versions of OpenCV.
- Timer development that is used in most of the modules created to manage timing within each module and in the main program.
- Memory module implementation to manage all the data provided by the user in order to identify him/her appropriately.
- Communication between boards using UART serial communication protocol in addition to XTEA encryption to secure the information transmitted from the Raspberry Pi to the SmartFusion.
- Testing of the encryption both in Python and in C code.
- Final integration of all modules.
- Final demo and correct functioning verification.

IV. RESULTS AND CONCLUSIONS

The final prototype used to perform final tests and demo is shown in Figure 5. In the illustration it is shown the SmartFusion board on the left side and the LoRa transmitter on the right side. This system accomplished all the main objectives set at the outset of the project, although some goals were achieved in a more satisfactory manner than others.

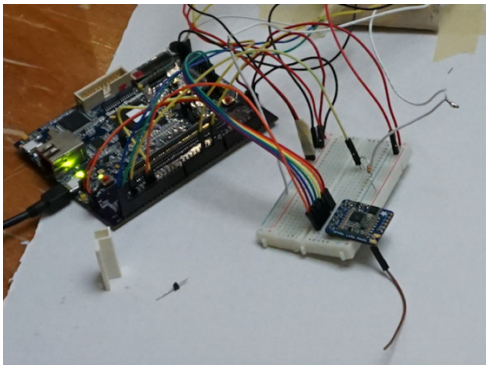


Figure 5: Final prototype.

The interconnection between components was deployed with success. The creation of the libraries let us access and have a better control over the device at a low level. The whole implementation to manage the full set of devices works effectively and robustly, given the absence of failures during the final stage of testing.

The encryption algorithms (RSA and XTEA) work as expected in LoRa gateway, SmartFusion and Raspberry Pi. However, it was desired to have a safer encryption, by adding some sort of communication protocol or a higher number of the key bits, (at least 256 bits, but the implemented RSA encryption uses 64 bits). Unfortunately, it was not possible due to time constraints and space for additional logical modules on the FPGA. Nevertheless, it serves as proof of concept.

The final integration was carried out successfully. The SmartFusion board is perfectly capable of managing all the processes being executed in the state machine, interrupts and algorithms. All the elements work effectively as expected, although on isolated occasions it experiments slight delays.

The face recognition algorithm works perfectly according to initial specifications. The more data provided (more images with different face expressions), the better the predictive model works. The algorithm has a correlation parameter between the actual face and model prediction that allows authentications with adjustable accuracy.

Even though all the goals set on the initial brainstorming and milestones were achieve, there are some aspects of the project that need to be improved. The system is designed to work under ideal conditions, i.e. in case a potential intruder disconnects the power supply or performs an electrical shock to the door bolt, the system does not have any kind of protection, and probably will stop working. This means that the system also requires electrical protections.

User experience constitutes the second aspect that was not addressed during project development. The web application graphic interface is too simple to be offered to the final customer. In addition, the warning message that is sent from the Raspberry Pi by email is independent from the server. It would be convenient to merge both features and create a smartphone application to be able to configure everything from a single place.

In conclusion, even though there are some aspects to improve, all the project goals were satisfactory accomplished, giving rise to a robust, effective and innovative authentication system based on credentials.

REFERENCES

[MR14] T. Miori and D. Russo. "Domotic Evolution towards the IoT". In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, Victoria, BC, 2014, pp. 809-814. doi: 10.1109/WAINA.2014.128

[HLMHSW16] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song and D. Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*. ACM, 2016. pp. 461-472.

[AHP06] T. Ahonen, A. Hadid and M. Pietikainen. "Face Description with Local Binary Patterns: Application to Face Recognition". In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037-2041, Dec. 2006. doi: 10.1109/TPAMI.2006.244

A mis padres, por su apoyo incondicional y por haberme brindado tantas oportunidades.

A Matthew, por su paciencia conmigo, los buenos consejos y las charlas sobre barcos.

List of contents

List of Figures

List of Tables

Abbreviations

IoT	Internet of Things
LPWAN	Low-Power Wide-Area Network
NFC	Near Field Communication
LAN	Local Area Network
EECS	Electrical Engineering and Computer Science
API	Application Programming Interface
BLE	Bluetooth Low Energy
LED	Light-Emitting Diode
PIR	Passive Infrared
SF	SmartFusion
RPi	Raspberry Pi
LBPH	Local Binary Patterns Histograms
GCC	GNU Compiler Collection
ASCII	American Standard Code for Information Interchange
MSS	Microcontroller subsystem
SR	Shift register
MMIO	Memory Mapped Input/Output
OS	Operating System
GPIO	General Purpose Input/Output
RFID	Radio Frequency Identification

Nomenclature

P	Encrypted message
X	Restored message
E	Public key
D	Private key
M	Message
i	Subkey index in Montgomery's algorithm
ei	ith digit of public key
n	Number of bits in public key
Z	Multiplier in Montgomery's algorithm
C	Cipher text

DOCUMENT I

MEMORY

Chapter 1

Background

As the number of smart devices around us grows more and more every day, new ideas come up and the implementation of their functionalities becomes necessary for applications to the real world. Therefore, new scenarios are occurring with new technologies, especially on the domestic sector, where many sensors have been developed to have an almost full control over the different house variables: temperature, lights, home appliances, etc. However, domestic security systems have not been developed enough in terms of technology availability in certain situations [MR14]. This project pursues the integration of a safe system to enhance both the home security – in terms of entry control – and door features. The main goal is to create a simple but effective way of getting into the house/apartment.

Consequently, work presented herein aims to offer a practicable solution to avoid the use of keys by using different IoT devices, that permit the entrance without using any physical key. All of this should be done maintaining the whole security related with the door at the same level of traditional locking systems. Therefore, not only the set of devices should be integrated correctly, but also it is necessary to build a secured network to transmit the necessary data, so encryption will be particularly essential in this application.

Monitoring home automation normally is based on Wi-Fi, connecting and sharing over the LAN network all the sensors data. In this work, different approaches are used to differentiate the prototype from current products made by competitors in the marketplace. Since security is the priority, wired connections are considered as default to connect devices. However, not all connections can be made wireless, especially if monitoring devices are connected to a server. LoRa modules are suitable for wireless connections due to its convenience in terms of efficiency, range and more features that will be described in following chapters.

In this section, a brief introduction will be given on how the work is carried out, objectives, motivation and the state of the question.

1.1 Motivation

House owners usually share the places where they live, either with roommates or as a family. Therefore, it results convenient to create a system capable of controlling the access. This project aims at people in communal living spaces such as apartments, shared houses or family detached homes who are looking for greater control over the security of entries by using a central system with credentials to allow access through doors as opposed to a simple key mechanism. To put it into practice, physical key must be replaced by another instrument easier to use and able to provide a straightforward entrance to the house while maintaining the place secured. There are two approaches used to obtain access: NFC and face recognition. These two procedures should be enough to achieve a reliable secured system that permits an easy access since smartphones have become an indispensable device and faces are inherent to people.

This project contributes to ordinary door locking systems by providing three main benefits. The first benefit is the added security thanks to the doors via two-factor authentication. Each door can only be unlocked either via something on you (i.e. NFC via a phone) or a part of you (i.e. face). This method surpasses using a physical key because it cannot be as easily duplicated or lockpicked, and is also convenient as the odds of losing a key are much greater than losing a smartphone. Obviously, faces always go with us. In addition, owners can check the status of the door in real time.

The second benefit is the freedom of access as long as there is a correct authentication. Each door can have both blacklists and whitelists to outright deny or accept people, and if someone is on neither, it is possible to request access remotely instead, much quicker than waiting for them to come home.

The third advantage is the ease of use of the system: simply by showing one's face will be enough to open the door. In case the owner is wearing scarf or any other accessory that may hinder the face recognition, permission will be granted by holding the phone near the reader. Both approaches take place immediately, saving time not only in the unlocking process of the door but also while looking for keys in deep pockets and purses.

1.2 Context

Since the methodology of the project is modified throughout the academic year, it is divided into two parts: Part I was completed during the Fall semester of 2017 as part of a University of Michigan EECS course, and carried out by three team members. Part II was developed during Winter semester 2018 by the author of this memory, as an independent study.

The first section was part of University of Michigan EECS 373 course: Introduction to Embedded Systems Design, and was developed and completed by Erik Liubakka, Kenneth Ozdowy and Cristian Gómez Peces. This section includes the major part of the hardware and software development that was integrated in the SmartFusion (microcontroller with an integrated FPGA that is used for the first part of the project as the "brain" that manages every smart device).

Part II involves all the implementation related to the face recognition module, which is based on a Raspberry Pi Model B, running computer vision algorithms implemented on the OpenCV API. This part was carried out as a directed study: 'EECS 499: Advanced Directed Study', and supervised by Matthew Smith, director of this work and also project instructor in the initial part of the project.

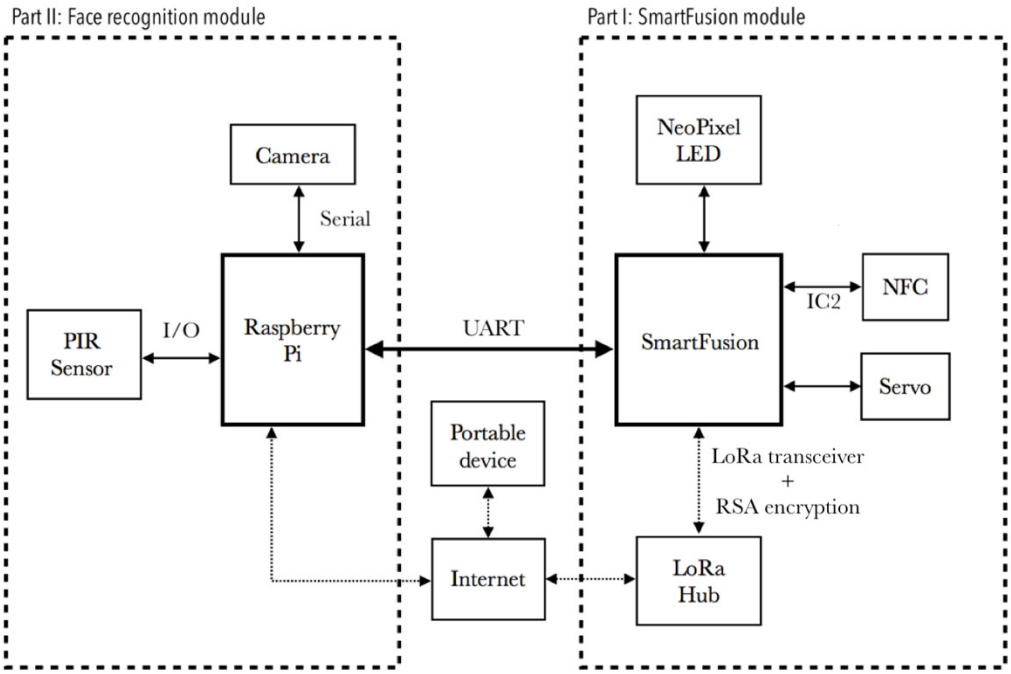


Figure 1.1: High-level scheme of the project elements

The overview scheme of project development can be seen in Figure 1.1. The SmartFusion module interfaces the NFC sensor, LoRa transmitters, LoRa hub and servo motor as lock and the NeoPixel LEDs. The second part, which includes the face recognition algorithm, PIR sensor implementation and final SmartFusion integration.

Since part I and part II follow the same structure, both have the same writing format: an introduction which states the motivation and goals of project part, a body that reports the implementation of both hardware and software and finally the conclusions, explaining the results of each part.

1.3 State of Art

The project is focused on home security, so it will be shown firstly the different technologies available for customers who want to install smart locks and cameras for face recognition in order to increase home security. In this section is analyzed these two elements in depth as well as their positioning in the marketplace.

1.3.1 Smart lock concept

Smart locks are devices that replace traditional deadbolts with electronically controllable ones which communicate with user's smartphones or the lock manufacturer's servers. These smart devices use IoT network architectures implemented in domotic systems as well as new characteristics that provide new ways of interacting with the device (wireless communication or authentication methods, for example).

Smart Locks are constituted by three elementary components: an electronical deadbolt installed onto an exterior door, a portable device capable of controlling the lock and a remote web server. The main architecture that smart locks usually use is shown in Figure 1.2. Smart locks themselves do not have a direct connection to the Internet. Instead, these locks rely on users' cellphones to act as an Internet hub that relays information to and from the manufacturer's servers whenever the phone enters BLE range of the lock [HLMHSW16].

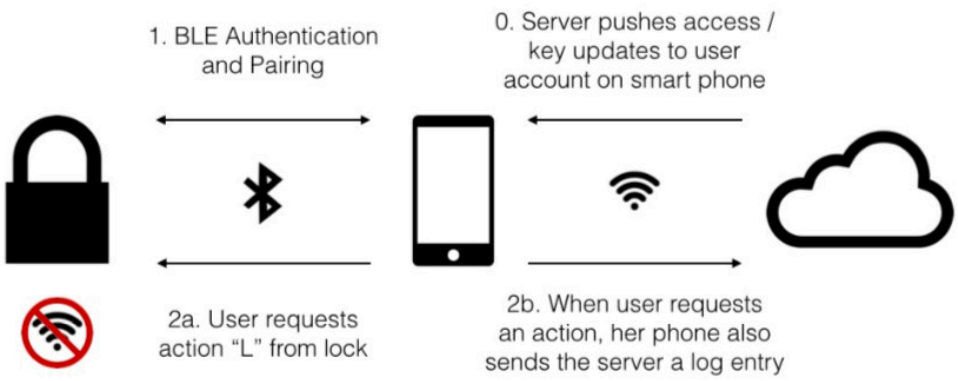


Figure 1.2: Smart lock connectivity [HLMHSW16].

In other words, smart devices connect with the smart lock at a local network and later on update the current status to the online server. There are some concerns over privacy of wireless communications, which still ranks high as a deterrent to wireless technology adoption. Bluetooth presents potential security vulnerabilities in the specification and in its implementation. For example, a security vulnerability could be entering into a invalid state, address validation or exposed keys. Therefore, to solve these problems, proper hardware implementation with strict configuration management needs to be enforced and some aspects of the Bluetooth protocol need to be redesigned before implementing it in safe applications such as home security [HM03].

1.3.2 Smart locks in industry

There are many different types of smart locks that implement different features to get keyless locking systems. Traditional locks usually come with a 10 digits pad so a PIN can be introduced. As technology advances, new authentication methods come out to the market, such as security tokens (for example NFC cards or smartphones with NFC integrated) or biometrics: fingerprint, hand geometry, eye scan or voice are used to verify the identification and grant access.

In addition to sensors and mechanical features, these devices can also incorporate additional characteristics based on digital algorithms. For example, multiple access solutions: the door can be unlocked by NFC, fingerprint or PIN code at the same time. Automatic locking or alarm integration usually come integrated in order to automatically lock a door after checking it is properly closed. It also alerts for burglary or vandalism in case they occur. Below it is shown three

different smart locks that are currently being sold in the marketplace: Samsung SHS-2920 EX Smart Door Lock, August Smart Lock, 3rd Gen technology and ZKTeco TL400B smart door lock.

Samsung SHS-2920 EX Smart Door Lock

The smart lock offered by Samsung. Among its specifications, it stands out the withstand to electric shocks (used by thiefs to override the proper functioning of the device), a resistance up to 60 degrees Celsius, automated locking and an alarm that sounds when several errors occur in the authentication access process. Figure 1.3 shows an illustration of device.

Third generation August Smart Lock

The smart lock developed by August includes a whole control of the door from anywhere (opens and locks it remotely), keeps track of who comes and goes, it can be easily integrate with assistants (such as Amazon's Alexa, Google Assistant or Siri) and can easily be implemented with the current deadbolt. Figure 1.4 shows an illustration of August's smart lock.

ZKTeco TL400B smart door lock

The smart door lock designed by ZKTeco appears in Figure 1.5. It offers various types of access for user's convenience: fingerprint, PIN key, NFC card and smart phone. It also includes automated locking, voice guider, smart alarm for low battery and illegal operations, normal open mode and external terminals to draw back-up power from a 9V battery.

Figure 1.5: ZKTeco TL400B smart door lock.

Figure 1.4: 3rd Gen technology August Smart Lock

Figure 1.3: Samsung SHS-2920 EX Smart Door Lock

1.3.3 Face recognition concept

Face recognition has been developed over the last 5 decades, since Woodrow Wilson Bledsoe started working on a system that could classify a set of photos based on horizontal and vertical coordinates using a RAND tablet[1] in the 1960s. Therefore, face features such as mouth, nose and eyes could be captured and stored in a database. Unfortunately, computer power was very limited to process all the information. In the 1970s Goldstein, Harmon, and Lesk came up with 21 specific subjective markers like lips thickness or hair color but as downside it still needed to be manually computed.

In 1988, Sirovich and Kirby began applying linear algebra to the problem of facial recognition. They created an approach to perform feature analysis on collections of facial images and relate them in a set of basic features: the eigenfaces approach. In 1991, Turk and Pentland expanded upon the Eigenface approach over facial classifiers by discovering how to detect faces within images. The eigenfaces method is one of the most well-known approaches to perform facial recognition and, in addition, it was the base for the more complex algorithms to come.

The first public large-scale application of face recognition took place in the super bowl 2002 and was used to report several criminals with little importance. However, the test was seen as a failure due to all false positive outputs. The system was not ready to work in large-crowds' environments. Facial recognition kept improving and being tested over the decade of the 2000s, and it started to be implemented for large-scale applications in social media (Facebook for example uses face recognition to tag people in photos automatically since 2010) or in airports in the last decade. For instance, in 2011 the government of Panama, partnering with the U.S. Secretary of Homeland Security, authorized a pilot program of FaceFirst's facial recognition platform in order to cut down on illicit activities in Panama's Tocumen airport [Fbhfr].

Face recognition is aimed to be a major standard for security as it gets in more in the day by day of citizens. Face recognition will play a significant role in security matter as well as daily life, and we are already starting to watch in smartphones, like 2017 Apple Iphone X, that has face recognition not only to unlock the device but also to store passwords, pay online, etc.

[1] The RAND Tablet was one of the first digital graphic device marketed as being a low-cost device. It came with a stylus that was considered a highly practical instrument.

Applications

As mentioned, in recent years face recognition has become significant in robust and reliable applications, such as implementations in smartphones (to unlock them or to grant safe purchases physically or online). Other fields within the facial recognition world that are currently being developed are detecting multiple enrollment, border crossing, surveillance and entertainment. Every field has a huge potential to explode the implementation of face recognition. For instance, a lot of time could be saved if the cameras installed at airports were able to allow the entrance to people whose faces match with passport ID. Or even in the check out when grocery shopping, like Amazon Go plan: computer vision, sensor fusion and deep learning are integrated in a conventional store that permits buying products without doing long lines. Companies such as Instagram or Snapchat may use facial recognition to apply filters as well as other companies to create virtual make-ups for clients. Another well-known and controversial application of face recognition is the criminal identification, in which all faces captured by public cameras are analyzed and compared with driver licenses using machine learning algorithms to identify suspects.

Algorithms

All methods used for facial recognition have a common denominator: they measure and extract the distances of different face features. The most common features are: the ridges between the eyebrows, distances between the cheekbones, the edges of the mouth, the distances between the eyes, length and width of the nose, the contour and profile of the jawline or the length and width of the chin [Iimcb]. The way all this information is processed constitutes the algorithm. There are many different types of algorithm for face recognition that have evolved along the computer vision history. *Principal Component Analysis* is a dimensionality reduction technique used for compression and face recognition problems. The face images are decomposed into various collections of eigenfaces and are reconstructed by applying weights to each of these eigenfaces. *Linear Discriminant Analysis* was developed by Fisher in 1930, from where comes the term 'fisherfaces', which is an appearance-based method successfully used for face recognition and that will be explained more in detail in chapter 7. *Neural Networks* is used in many applications like pattern recognition problems, character recognition, object recognition etc. Neural networks main objective is to capture the complex class of face pattern and it is used in some of the latest cutting-edge products due to its ability to capture

more details[2]. *Graph Matching* is employed to find the closest stored graph by estimating a set of features. Finally, in the *Hidden Markov Model*, face is divided into regions that can be associated with the states of hidden Markov model like the eyes, nose, mouth, etc. Recognition rates with this approach are around 95% successful based on 2D models [VK15]. These six approaches are not the only ones but within the most common. We also have *Support Vector Machine* or *Local Binary Patterns*, which will be explained more in depth in Chapter 7 since it is the method implemented for the face recognition task in this project.

[2] Devices running face recognition based on neural networks usually need higher processing specifications.

Part I

Access Controller

Chapter 2

Introduction

Chapter 2 presents a brief introduction to the first part of the project: the SmartFusion module. Below it is explained the basic functioning of the system, main objectives, feasibility and considerations taken into account in the initial brainstorming, components list and their descriptions, the division of labor and methodology.

This initial part was carried out by three team members: Erik Liubakka, Kenneth Ozdowy and Cristian Gómez Peces as a part of a required project for a University of Michigan Electrical Engineering and Computer Science course during the fall term: EECS 373 Design of Microprocessor Based Systems. Half of the course was dedicated to learning the hardware and software tools using Libero and SoftConsole to implement memory mapped IOs, interrupts, timers, serial interfaces, ADCs and DACs. The rest of the semester was oriented to the project development.

During the second part of the course, teams were created according to what the team members desired to do, based on ideas exposed by themselves in class. In particular, the team of this project came up with the idea of developing a system capable of granting home access with no keys. The initial idea was to create a network capable of warning the owners about new access requests as well as granting access permission remotely to guests. The concept of the project is explained in detail in the high-level description section.

If desired, all the code (from both part I and II) can be found in the annexes at the end of the market study document. Modules are structured by parts, including source code, headers, main programs, python code, etc.

2.1 High level description

The goal of development of part I is to create an automated lock that opens the door with a key-less control system. The door can only be unlocked using an NFC sensor that is hooked up to the microcontroller integrated in the SmartFusion SoC (System on Chip). At the same time, the SmartFusion is connected to the internet through a wireless communication thanks to LoRa transceivers and hub. The system can have both blacklists and whitelists to outright deny or accept people. To be constantly updated, it is connected to a webserver implemented in the LoRa gateway, using the interfaced LoRa transmitter (which allows wide range communications at a low power consumption) and RSA as public key encryption method. The whole process is shown in figure 2.1. With this system, remote access requests by the owners and door status checks in real time will be possible.

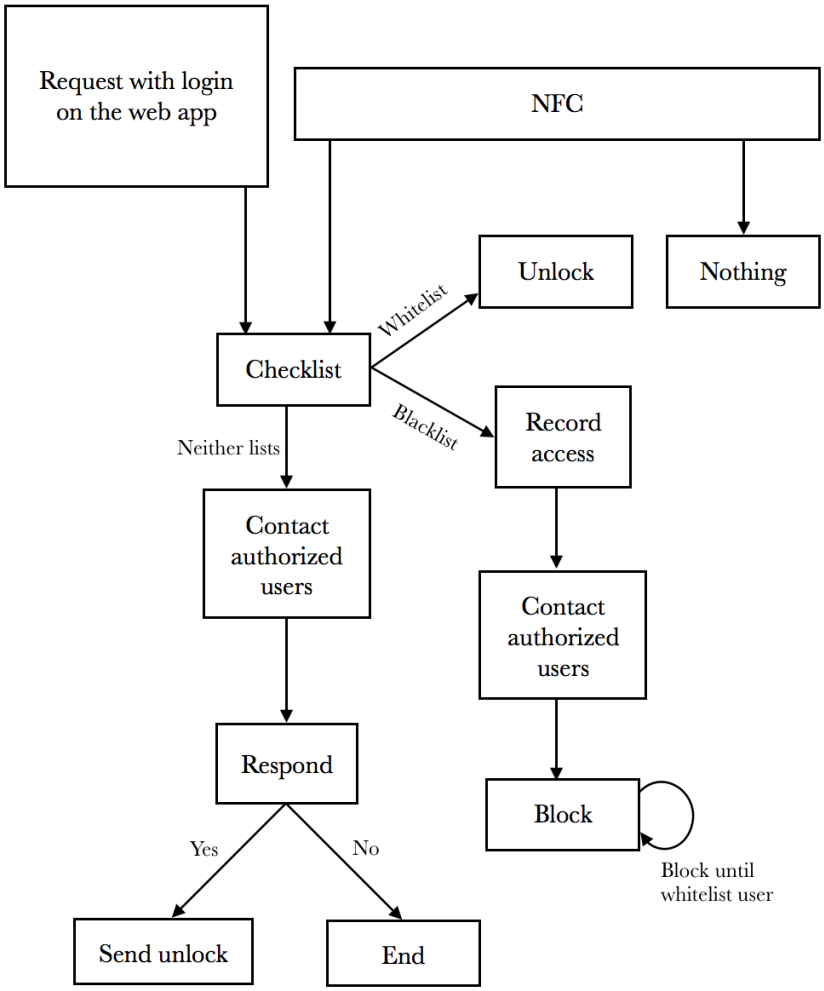


Figure 2.1: Unlocking process via NFC

The top-level design consists of a central hub with multiple nodes connected. The hub allows users to communicate with the lock via a web interface. The gateway is wanted to store what is necessary, acting as a control station while each of the locks do the security part, making it much harder for someone to spoof access rights. The connection to the SmartFusion is done via the LoRa transceiver and all the communications are encrypted using an RSA module implemented in the SmartFusion fabric. The information downloaded from the server is decrypted in the FPGA and processed by the Cortex M3. When the NFC sensor detects an electromagnetic field, it encodes the ID of the device/tag and sends it so the Hub, where it is compared with the blacklist and whitelist and depending on whether the ID is in the whitelist or blacklist, it will outright allow or deny the access, respectively. In the particular case in which the system deals with a non-registered target, it will pause the locking system until any of the owners approves or denies the transmitted request. Figure 2.2 shows a descriptive illustration of the identification process administration.

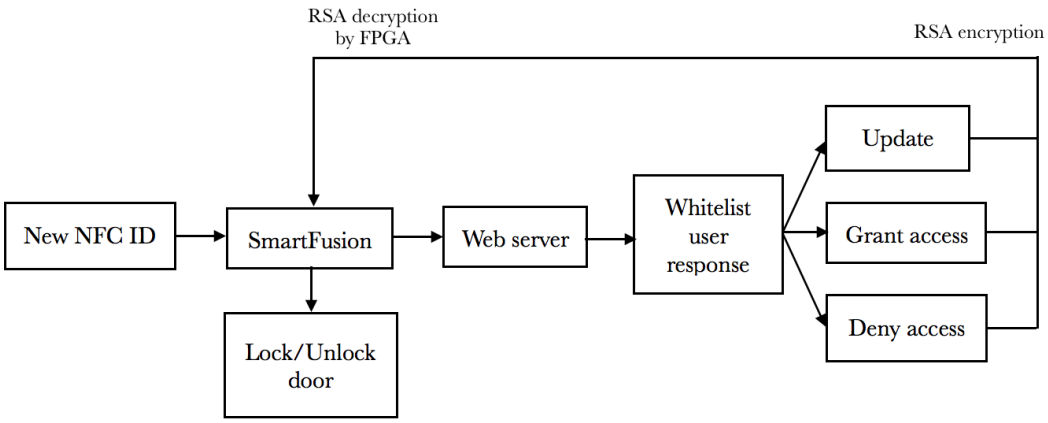


Figure 2.2: Administration of NFC authentication.

NeoPixel LEDs will be used to provide visual feedback for the locks. There are different colors depending on the current state (waiting for response, access granted, access denied, and an open door).

This approach provides three benefits mentioned before in the introduction: more security (higher control over the entries), freedom of access (it is possible to enter even with no keys; simply by requesting access to yourself or to your family/roomates via online) and ease of use: no more keys that need to be found

inside the pursue. With a simple swipe of your smartphone or NFC card the system will grant access after processing the ID.

2.2 Feasibility and considerations

In the initial brainstorming, the approaches proposed included a few more features to implement, as well as different ways and additional devices to achieve certain objectives of the project. For instance, the gateway that initially was going to be integrated was the Intel Edison, but the Dragino gateway featuring LoRa technology was finally chosen due to its ease of use when integrating it with LoRa transmitters. It also includes additional features such as SMS sending capability that could be useful to communicate with the owners. Other devices and sensors were intended to be integrated in the system, such as fingerprint sensors, but due to the need capacitive detectors and time constraints the team decided to focus on the main features that the system required: tags identification, wireless communication and encryption.

The project had three main milestones to decide which elements should have the highest priority and which ones were secondary objectives. The final project proposal should be approved by the EECS 373 theory professor and Matthew Smith, lab instructor and director of this project. In the first proposal, the Intel Edison board was thought to work well in our system, as well as having an Arduino in charge of the locking process. After discussing this approach, the LoRa gateway was chosen instead of the Edison and the SmartFusion was chosen to be the "brain" of the whole system. Not only do the SmartFusion board provides more computability power, it also permits adding features otherwise impossible for the Arduino. Firstly, we can implement our secure communication via RSA fully in hardware. The change also allows more practical implementation of class material as opposed to prebuilt Arduino libraries. In the final proposal the team decided to simulate cabinet locks, instead of using regular door locks, with the servo turning a piece of metal that will prevent the door from opening.

LoRa transmissions, NFC sensor interface, and RSA encryption implementation were enough sophisticated features to be integrated. Therefore, it was decided to focus on those elements to get the basic access controller: a functional system based on a secure authentication with safe and reliable connections between the SmartFusion and the webserver-based database that stores the information. On the other hand, features such as the additional authentication

device (fingerprint sensor) or SMS sending were assigned as secondary objectives that the system could work without them.

2.3 Objectives

The main development goals can be summed up in three main objectives. Firstly, it should implement the essential devices that need to be working after the integration (NFC, Lora transmitters and Hub, and servo). Secure the system using RSA encryption and ensure the communication is safe (so the system can ignore other possible LoRa devices in use next to our system location). Finally, it should integrate all the components with a low rate of failure: NFC, LoRa, RSA encryption, NeoPixel, servo and the web server should work coherently (figure 1.3).

Objective	Description
Implementation of components	Every component integrated within the system should function as it is supposed to. The implementation of the libraries should be complete and reliable.
Security of the system	Every single communication should be secured, i.e. the wireless transmission using LoRa should be protected from interception and the information should not be accessible in any case.
Integration of every component	The whole system should work properly. The microprocessor should handle the requests, data sending and communication between the different parts of the system. The components should work accordingly, and the microprocessor should transmit inputs and outputs where necessary.

Figure 2.3: Main objectives

The system has more specific goals within these three main objectives that depend on the component developing stages:

- Interface NeoPixel: it should be able to display different colors (green, yellow, blue and red) that will provide useful feedback (field detected, door opened, closed or blocked).
- Interface NFC: it should be able to read NFC tags/smartphones ID and send them to the Cortex M3.

- Make sure the parameters for the servo are appropriate to lock and unlock the door (it should turn 90º).

- Interface LoRa module: it should be able to communicate wirelessly by using the LoRa transmitter and the LoRa gateway.

- Achieve a reliable encrypted connection via RSA. First it is needed to implement it in Python and see if it works. Testing and final implementation in Verilog comes afterwards. The state machine implemented should consider the memory available so it possible to allocate other resources in the fabric.

- Connect LoRa transmitter to gateway and send data over the 915 MHz bandwidth.

- Set up a server in the LoRa hub: use the tools provided by dragino to set up the gateway and program the web server with python.

- Design a stylish but functional door and frame for demo the prototype using SolidWorks.

2.4 Division of labor

In order to achieve the goals stipulated in the initial brainstorming (see section 2.5: methodology) the team distributed the workload as follows:

- Kenneth Ozdowy (33.33%). In charge of the NeoPixel implementation. Also worked with Cristian on the Lora transmitters (in LoRa transmitter drivers' development) and developed the web server using Python. Additionally, Kenneth offered support to Erik in the RSA encryption implementation.

- Erik Liubakka (33.33%). Worked on the NFC implementation with Cristian (specifically in the high-level functions for the NFC library). In charged of the RSA implementation, Erik implemented the RSA algorithm in Python and in Verilog with Kenneth's support.

- Cristian Gómez Peces (33.33%). Worked on the NFC implementation with Erik. Cristian developed the NFC drivers necessary to interface the PN532. Worked also on the LoRa implementation with Kenneth, specifically on the communication between transmitters and hub, and

gateway set up. In charge of the door and frame design for the exposition.

All the team members worked together on the final integration of the project. They also devised and created the poster for the exposition.

2.5 Components descriptions

There are 6 main components to integrate in this submodule of the project: the SmartFusion, the NFC sensor, the contact switch, the NeoPixel, LoRa transmitter and LoRa gateway.

2.5.1 SmartFusion

It is the heart of the system: controls and manages all the processes that are taking place during the execution of the state machine. Figure 2.4 shows an illustration of this SoC. It integrates an FPGA Fabric, ARM Cortex-M3 processor and a programmable analog circuitry.

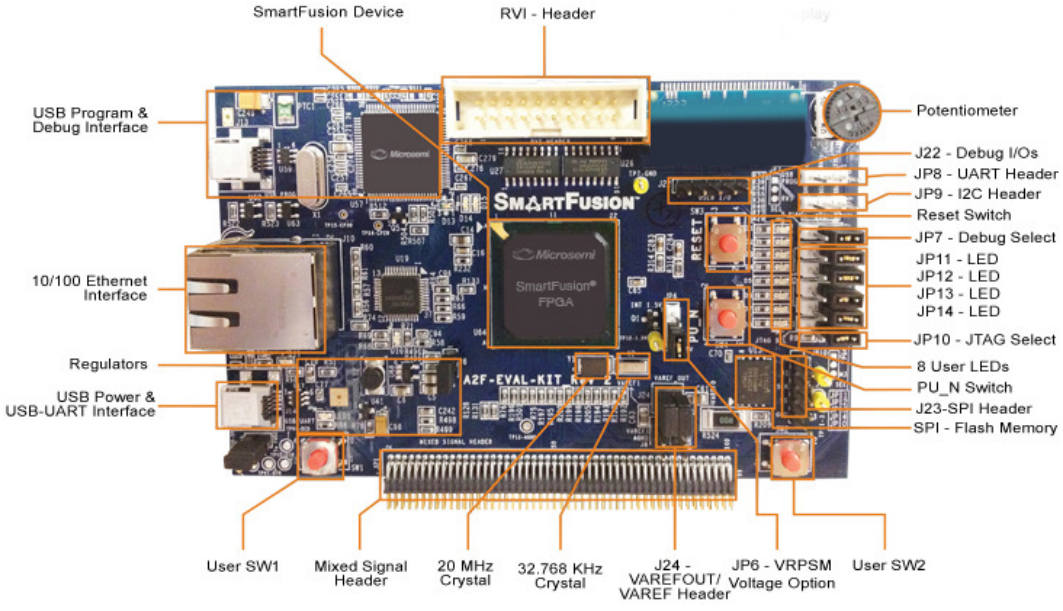


Figure 2.4: SmartFusion SoC.

These chips aim at systems that require more flexibility than traditional fixed-function microcontrollers. Figure 2.4 shows the three parts integrated within the board, which are analyzed below (this information is provided by the manufacturer).

1) Microcontroller Subsystem (MSS). These are the specifications of the microcontroller subsystem integrated in the SmartFusion:

    - Hardware industry-standard 100 MHz, 32-bit ARM Cortex-M3 CPU
    - Multi-layer AHB communication matrix with up to 16 Gbps throughput
    - 10/100 Ethernet MAC with RMII interface
    - Two of each: SPI, I2C, UART, 32-bit timers
    - Up to 512 KB flash and 64 KB of SRAM
    - External memory controller (EMC)
    - 8-channel DMA controller
    - Up to 41 MSS I/Os with Schmitt trigger inputs

2) FPGA Fabric, which includes:

    - Based on Microsemi's proven ProASIC3 architecture
    - 60,000 to 500,000 system gates with 350 MHz system performance
    - Embedded SRAMs and FIFOs
    - Up to 128 FPGA I/Os.

3) Programmable analog. This circuitry allows to sample and process signals as well as convert digital signals into analog signals. It consists of:

    - High-performance analog signal conditioning blocks (SCB) with voltage, current and temperature monitors
    - Analog compute engine (ACE) offloads CPU from analog initialization and processing of analog-to-digital conversion (ADC), digital-to-analog conversion (DAC) and SCBs
    - Integrated ADCs and DACs with 1 percent accuracy
    - 12-/10-/8-bit mode ADCs with 500/550/600 Ksps sampling rate
    - Up to ten 15 ns high-speed comparators
    - Up to 32 analog inputs and 3 outputs

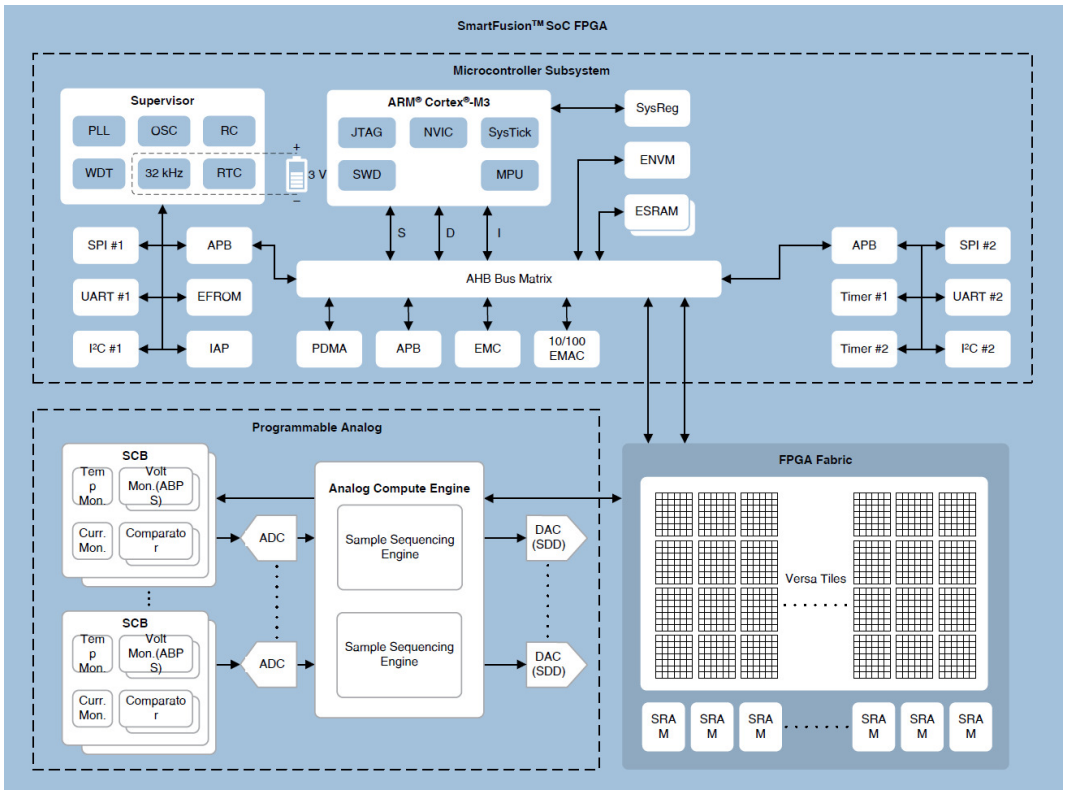


Figure 2.5: Integrated parts of the SmartFusion.

2.5.2 PN532 NFC sensor

The PN532 is one of the most popular NFC chips (Figure 2.6) that allows to read and write NFC tags and everything that acts as NFC tag (smartphones included). The near-field communication takes place between two electronic devices that should be within 4 cm to each other. The near field communications offer connections at a low rate but with a simple setup that makes it suitable for wireless connections. It is also a safe system since the maximum distances to make it work are around 10 cm. In this case, it is be used to read identity keycards. The main advantage of using the PN532 model is its flexibility: it is possible to use 3.3V TTL UART at any baud rate, I2C or SPI to communicate with it. This chip is also supported by libnfc[3], so there is a lot of documentation and a supportive community to develop software using this sensor.

[3] Libnfc is the first free, platform-independent, low level NFC SDK and Programmers API. Its development status is mature (after alpha and beta versions), so it is reliable.

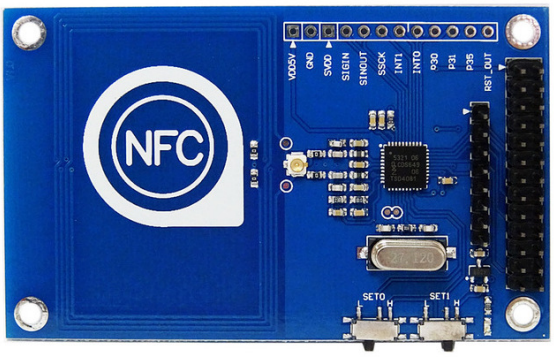


Figure 2.6: PN532 NFC sensor implemented.

2.5.3 Contact switch

A magnetic contact device is attached to the door and door frame, so the SmartFusion can know whether the door is open or not. This allows a better control over the door status. The use of this device (Figure 2.7) is convenient due to its simplicity to implement and to the useful information it provides (if the door is never closed the system will be able to notify the owners).

Figure 2.7: Contact switch sensor.

2.5.4 LoRa transmitter

The LoRa transmitter is a key device in the project since thanks to its implementation it is possible to connect the SmartFusion to the internet with a low power consumption. Not only stand out the LoRa transmitters because its low energy consumption but also due to the long ranges the it is capable to reach (from 2km to 20km depending on the antennas). In addition, it offers four different

frequencies to work with. Below it is shown the main features of these transmitters (provided by seller):

- Packet radio with ready-to-go Arduino libraries: example to implement in order platforms.
- Use a simple wire antenna or SMA radio connector
- Module based in SPI interface
- +5 to +20 dBm up to 100 mW Power Output Capability (power output selectable in software)
- ~100mA peak during +20dBm transmit, ~30mA during active radio listening.
- Range of approx. 2Km, depending on obstructions, frequency, antenna and power output. Up to 20km with directional antennas.

Figure 2.8 shows the LoRa transmitter that was integrated. Note: it is necessary to solder the headers in order to connect it to the board.

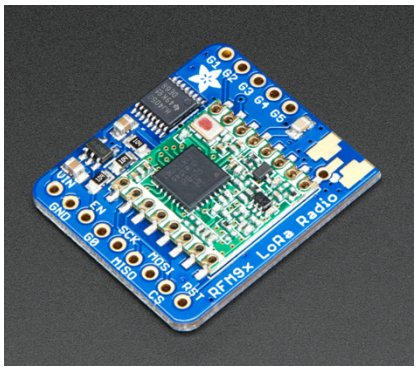


Figure 2.8: LoRa transmitter.

2.5.5 LG01-S: Gateway featuring LoRa technology

The LG01-S is an open source single channel LoRa Gateway. It permits bridging LoRa wireless network to an IP network via WiFi, Ethernet, 3G or 4G cellular (see datasheet in appendix F for more information). A workflow diagram and an image of the gateway are shown in figures 2.7 and 2.8.

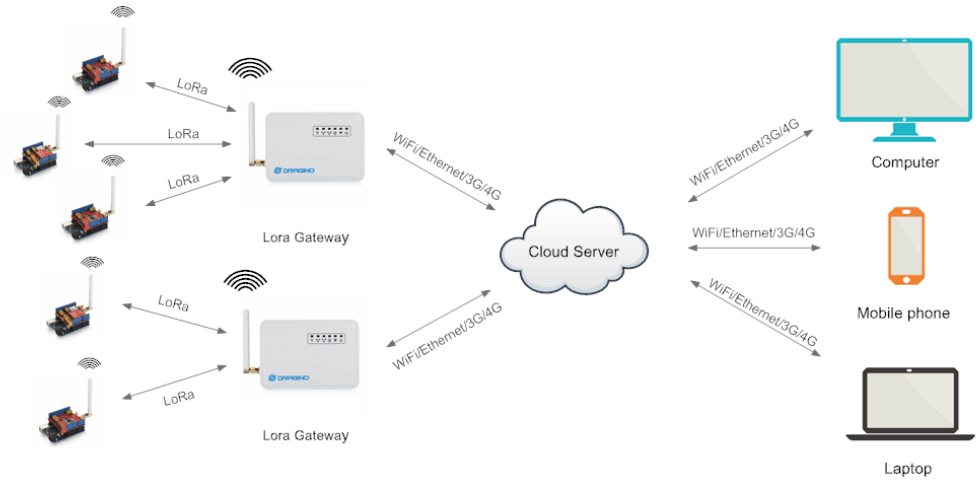


Figure 2.9: LoRa transmission workflow.

The LoRa gateway (sometimes called as "dragino" in this document due to its manufacturer) allow the SmartFusion to be connected to the web server through the gateway-transmitter connection.

Figure 2.10: LG01-S (LoRa gateway).

2.5.6 NeoPixel LED

Normally these LEDs come in packs of connected LEDs one after each other, so the different colors that they produce can be combined producing eye-catching colorful effects. This project particularly required only one of these LEDs. As can be deduced, the control is simpler than with a row of LEDs. However, this

component does not have a standard serial connection, but its own protocol. It is possible to set the color of each LED's red, green and blue component with 8-bit PWM precision (24-bit color per pixel). The LEDs are controlled by shift-registers that are chained up down the strip and only 1 digital output pin is required to send data down. The PWM is built into each LED-chip so once you set the color you can stop talking to the strip (see datasheet for more information). Figure 2.9 shows a figure where appears a set of these NeoPixel LEDs.

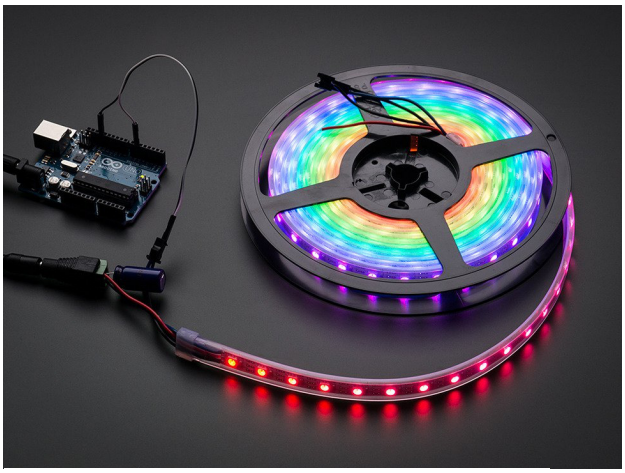


Figure 2.11: Set of NeoPixel LEDs.

2.5.7 Hitec HS-422 - Deluxe Standard Servo

This high-torque standard servo can rotate up to 180 degrees (90º to each side). This device was previously interfaced in one of the lab of EECS 373 so it was only required to adapt this particular implementation. Figure 2.10 shows a picture of the servo used to lock the door.

Figure 2.12: Hitec HS-422 - Deluxe Standard Servo

2.6 Resources provided by the University of Michigan

The University of Michigan contributes to the development of this project providing necessary software and hardware. Since the beginning of the project, one workstation is assigned per team. Each workstation runs Windows with multiple tools that can be used to create code. In this case, the programs Libero and SoftConsole are essential to develop all the implementations. Libero is used to program the FPGA, providing the hardware needed to connect all the devices to the SmartFusion. The hardware description language used for that purpose is Verilog. On the other hand, SoftConsole is used for software development; it is a powerful tool since it incorporates multiple tools for debugging. Some of the components are acquired by the university and others by the team members. The NFC sensor, NeoPixel LEDs, contact switch, and LoRa transmitters were acquired in online stores, whereas the university of Michigan provided the LoRa gateway, servo, and the SmartFusion board.

2.7 Methodology

During the first part (Fall 2017), the work is carried out in team and therefore, a version control tool is needed to share, edit and create code. GitHub was chosen for this task because the members of the team were familiar with it. In addition, it gathers all the features we need to work.

Firstly, the libraries that interface the devices were created and tested, performing real application tests. Once the implementation was completed, the next task for a certain device was assigned, following the time-schedule raised in the initial milestones and brainstorming. Finally, every member of the team participated in the final integration of the components. Figure 2.11 shows an illustration of the Gantt diagram that shows time estimations and milestones that were followed during the second half of the Fall term.

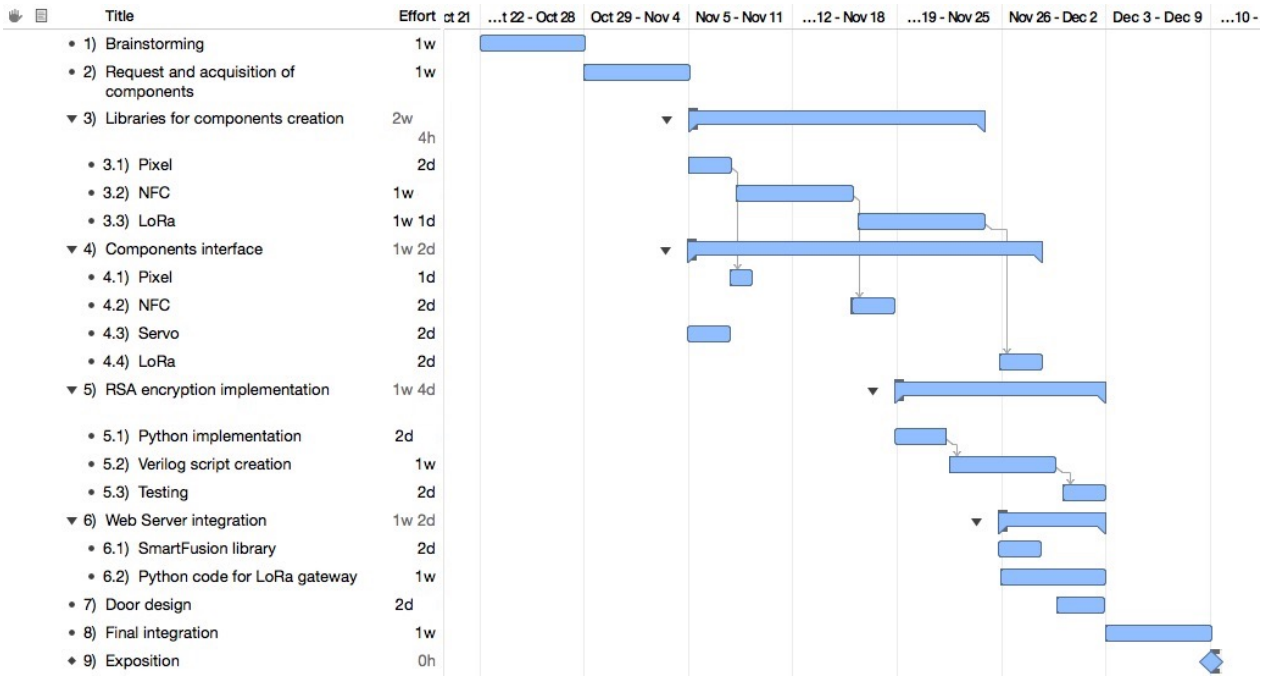


Figure 2.13: Planification for the first part of the project.

Chapter 3

Hardware implementation

Chapter 3 explains how connections in hardware were made, showing the different FPGA modules and buses used to interface all the components. Every component has different ways to be interfaced depending on their serial protocols and interrupt triggering. This section will go over the specifications of the component (I/Os, serial ports, interrupts, etc.), logical modules integrated in the fabric, Verilog code to implement them, and the RSA encryption algorithm development integrated into the FPGA.

3.1 MSS configuration and APB3 Interface

The MSS configuration is shown is Figure 3.1.

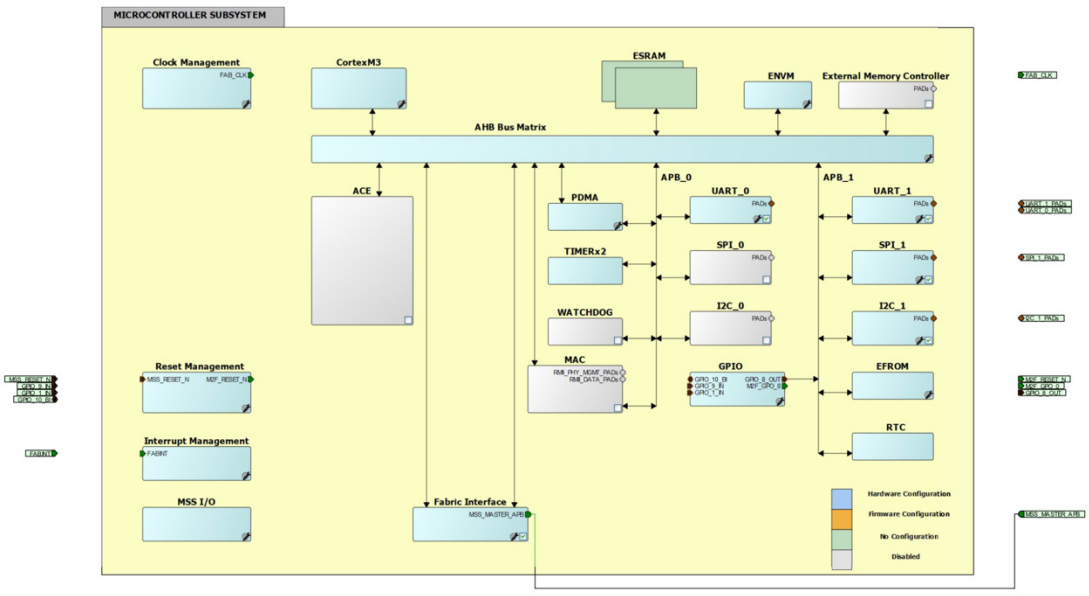


Figure 3.1: MSS configuration.

There are two interrupts that need to be handled. GPIO1 is assigned to the NFC interrupt pin whereas GPIO9 handles the interrupt triggered by the LoRa transmitter. UART_0 is enabled to communicate with the computer for debugging and flashing software. SPI_1 is enabled to interface the LoRa transceiver and I2C_1 to interface the NFC sensor.

According to the APB4 interface datasheet, the Advanced Peripheral Bus is part of the Advanced Microcontroller Bus Architecture protocol family. It defines a low-cost interface that is optimized for minimal power consumption with reduced interface complexity. The APB protocol has two independent data buses, one for read data and one for write data. The buses can be up to 32 bits wide. Because the buses do not have their own individual handshake signals, it is not possible for data transfers to occur on both buses at the same time. An example of a write transfer is shown in Figure 3.2.

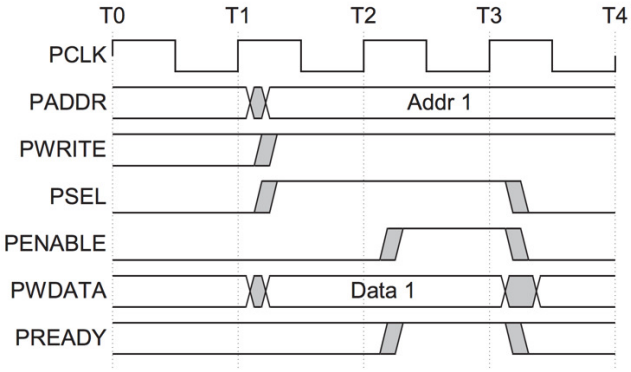


Figure 3.2: APB3 write transfer.

For any type of operation (write or read), bus works as follows. The transfer starts with address PADDR, write or read with PWDATA or PRDATA and select signal PSEL, triggered at the rising edge of PCLK as the action is requested. In the next positive clock edge, signal PENABLE and PREADY are set to high to proceed with the write/read. Once the information has been transferred, the signal PENABLE is disabled as well as PSEL as long as there is no transfer to the same peripheral.

Finally, the signal PSLVERR can be used to indicate an error condition on an APB transfer. Error conditions can occur on both read and write transactions.

3.2 Verilog: HDL files implementation

Verilog is a hardware description language which can describe the operation of a gate-level schematic of a circuit rather than drawing it. Its structure is very similar to C in many ways —the same style of comments, the same operators and similar control structures, but with one main difference: its execution is inherently parallel, which means that events will not happen sequentially, but at the same time.

Each component should have its own HDL file in order to describe how is it going to behave: what is the protocol of data transfers, connections to other modules, instances connected to the APB bus, whether or not it has interrupts (and determine the type of interrupt handler) and the GPIOs that are necessary to interface it correctly. The HDL files are organized in hierarchy, so at the low level only the modules are the defined and it is possible to declare instances in the apb3 module later. In other words, the modules NFC.v, servo.v, neopixel.v and RSA.v define the behavior of the essential elements connected to the board whereas apb3_interface.v declares the all the instances and assigns the required connections to interface every of the components. Therefore, apb3_interface.v constitutes the top level files that connects most of the modules. Other components of the system are interfaced through the MSS, enabling the UART, SPI and I2C instances so it is possible to read/write them using the MSS drivers incorporated in the pre-built libraries.

Last but not least, it is required to assign peripherals to the connections that were set on the .hdl file. Attributes menu allows the allocation of the GPIO connections. Figure 3.3 shows a screenshot of the peripherals assigned to the wired connections.

	Port Name	Group	Macro Cell	Pin Number	Locked	Bank Name
1	FABINT		ADLIB:OUTBUF	G4	☒	Bank5
2	GPIO_0_OUT		ADLIB:OUTBUF_MSS	V1	☒	Bank4
3	GPIO_1_IN		ADLIB:INBUF_MSS	R3	☒	Bank4
4	GPIO_4_IN		ADLIB:INBUF_MSS	AA1	☒	Bank4
5	GPIO_8_OUT		ADLIB:OUTBUF_MSS	T3	☒	Bank4
6	GPIO_9_IN		ADLIB:INBUF_MSS	V3	☒	Bank4
7	I2C_1_SCL	I2C_1_PADs	ADLIB:BIBUF_OPEND_MSS	U20	☒	Bank2
8	I2C_1_SDA	I2C_1_PADs	ADLIB:BIBUF_OPEND_MSS	V22	☒	Bank2
9	MSS_RESE...		ADLIB:INBUF_MSS	R1	☒	Bank2
10	NP_OUT		ADLIB:OUTBUF	F1	☒	Bank5
11	RAM_RESET		ADLIB:TRIBUFF	K19	☐	Bank1
12	SERVO_OUT		ADLIB:OUTBUF	J6	☒	Bank5
13	SPI_1_CLK	SPI_1_PADs	ADLIB:BIBUF_MSS	AA22	☒	Bank2
14	SPI_1_DI	SPI_1_PADs	ADLIB:INBUF_MSS	V19	☒	Bank2
15	SPI_1_DO	SPI_1_PADs	ADLIB:TRIBUFF_MSS	T17	☒	Bank2
16	SPI_1_SS	SPI_1_PADs	ADLIB:BIBUF_MSS	W21	☒	Bank2
17	UART_0_RXD	UART_0_PA...	ADLIB:INBUF_MSS	U18	☒	Bank2
18	UART_0_TXD	UART_0_PA...	ADLIB:OUTBUF_MSS	Y22	☒	Bank2
19	UART_1_RXD	UART_1_PA...	ADLIB:INBUF_MSS	W22	☒	Bank2
20	UART_1_TXD	UART_1_PA...	ADLIB:OUTBUF_MSS	V20	☒	Bank2

Figure 3.3: Attributes.

3.3 Component specifications

In this section it is explained in detail the hardware that each component has, the connections that it requires, and the modules implemented in the fabric to interface it. Figure 3.4 shows the connections between hardware modules. As can be seen, most devices are interfaced using MSS block or the APB3 interface.

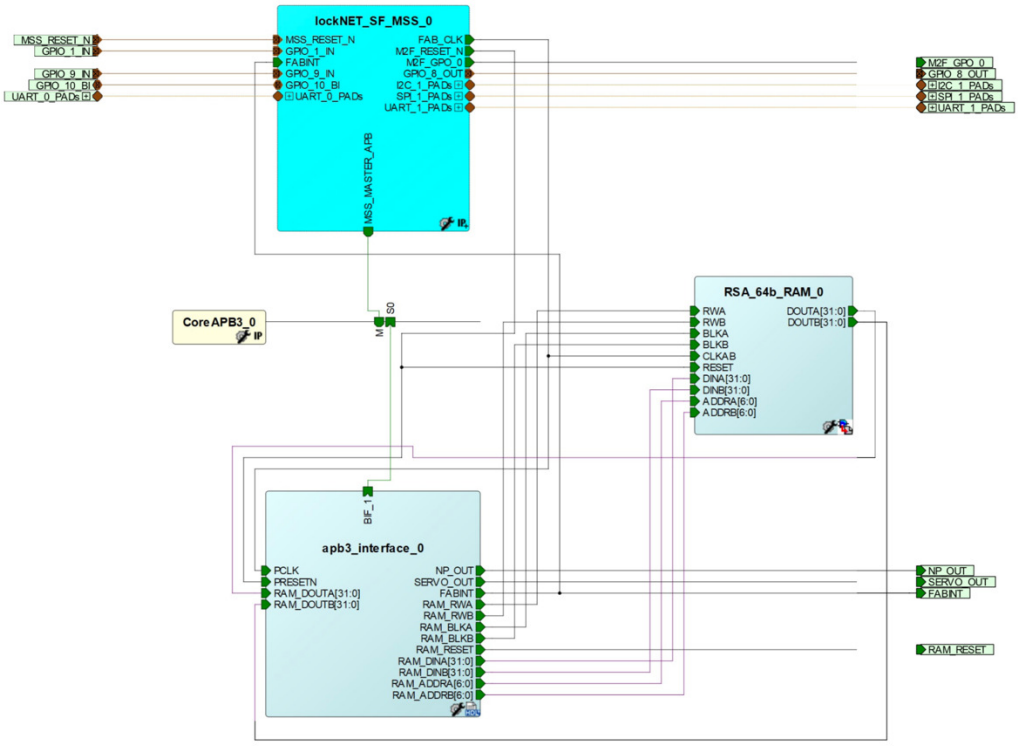


Figure 3.4: Hardware connections.

3.3.1 NFC sensor

The NFC offers multiple ways to be interfaced. According to the datasheet, it supports SPI, I2C and UART serial protocols, so it has 4 pins for the SPI port (SCLK, MOSI, MISO and SS), two pins for the IC2 port (SDA and SCL) and 2 pins for the UART serial port (Tx and Rx), in addition to the power supply connections. The serial protocol used in the drivers was I2C due to all the documentation that is available (SPI and I2C are the most common serial protocol used to create software libraries of the NFC sensor). VCC is connected to +5V and there is also a pin to trigger interrupts, so the host controller knows when the data is ready to be read.

To sum up, the NFC is implemented in the fabric as a module that checks every clock edge the state of the interrupt pin. All the read and write transfers are performed through the I2C serial bus in the MSS, and all this is driven by software.

3.3.2 LoRa transceiver

LoRa technology uses SPI as serial protocol to communicate with microcontrollers. Therefore, it includes the four pins for the serial connections, VCC and GND, reset pin, and a peripheral for interrupts triggering. The transmitter needs no module in the fabric since the SPI1 instance is enabled from the MSS and it is driven from software. It also needs an interrupt that is assigned to a GPIO, whose handler is written in software. The transmitter was soldered with jumpers, so it could be implemented in a breadboard. Finally, the antenna is basically a ten centimeters cable that is soldered to the transceiver.

3.3.3 NeoPixel LED

The NeoPixel LED uses the APB bus interface to communicate with the microcontroller. MMIO is used to set the pixel value in a register, then custom driver code to send out the color properly as NeoPixel LEDs use a self-clocking signal to transfer data. The color of the LED is firstly set by software and finally applied to hardware using built hardware drivers.

3.3.4 Servo

The angular position is controlled by applying a variable PWM signal. With a fixed period, the servo receives a threshold of the time it should high during that fixed period (500,000 clock cycles). This threshold is defined in software and applied using used MMIO and two defined values to lock (45500 clock cycles) and unlock (2300 clock cycles).

3.4 RSA encryption implementation

The FPGA module implemented to cipher and decipher the messages sent through LoRa was an RSA public key encryption/decryption system. Figure 3.5 shows a usual encryption/decryption workflow of an asymmetric public key system.

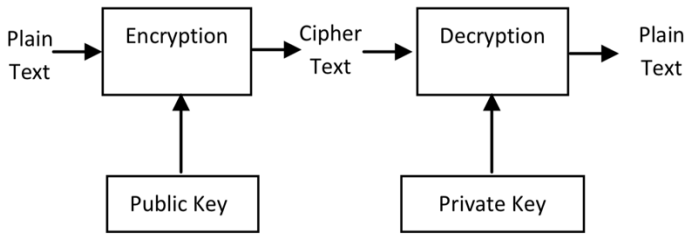


Figure 3.5: Public key cryptography [KP11].

It was implemented the modular exponentiation that RSA is built on by using Montgomery's Modular Multiplication algorithm [HSS00]. To ensure the algorithm was understood, it was firstly written a Python script implementing it:

```
1.  import sys
2.
3.  def MonMult(A, B, n):
4.      P = [0]*65
5.      q = [0]*64
6.      assert(A < n)
7.      assert(B < n)
8.      for i in range(0, 64):
9.          q[i] = ((P[i] & 0b1) ^ (((A >> i) & 0b1) & (B & 0b1)))
10.         P[i + 1] = (P[i] + (((A >> i) & 0b1) * B) + (q[i] * n)) / 2
11.     if P[64] >= n:
12.         result = P[64] - n
13.     else:
14.         result = P[64]
15.     return result
16.
17. def ModExp(C, d, n):
18.     Z = [0]*65
19.     result = [0]*65
20.     residue = (2 ** (64*2)) % n
21.     result[0] = MonMult(residue, 1, n)
22.     Z[0] = MonMult(residue, C, n)
23.     for i in range(0, 64):
24.         result_temp = MonMult(result[i], Z[i], n)
25.         Z[i+1] = MonMult(Z[i], Z[i], n)
26.         if ((d >> i) & 0b1):
27.             result[i + 1] = result_temp
28.         else:
29.             result[i + 1] = result[i]
30.     final = MonMult(1, result[64], n)
31.     return final
32.
33. def main():
34.     message = 0xdcbaabcdabcddcba
35.     pub = 0x10001
36.     private = 0x44de026cabdb311
37.     modulus = 0xeda515ef24029417
38.
39.     correct = (message ** pub) % modulus
40.     cypher = ModExp(message, pub, modulus)
41.     decypher = ModExp(correct, private, modulus)
42.     print(hex(message))
43.     print(hex(cypher))
44.     print(hex(decypher))
45.     print(hex(correct))
46.
47. if __name__ == '__main__':
48.     main()
```

Once it was successfully tested, it was designed a module in Verilog that implemented the Montgomery algorithm, and then designed a state machine utilizing this module to complete the encryption or decryption of a 64-bit message.

To use MonMult() in ModExp(), the input operands A (residue of performing the modulo operation) and B (message) for ModExp() are first transformed to the M-residue domain by performing A· R mod M and B · R mod M respectively. This is performed in step 1 in algorithm below, where R^2 mod M is precomputed and unchanged throughout the whole cryptographic processing using the same key. Thus, the product will carry an extra R modulo M. This product can be reused as the operand in the next Montgomery modular multiplication. The final product is transformed back to normal integer by eliminating the extra R mod M [HSS00]. Thus, ModExp() function becomes:

0. Compute $P = X^E mod M$, $\sum_{i=0}^{n-1} e_i 2^i$, $e_i \in 1, 0$
1. $P_0 = MonMult(1, R^2 mod M)$

   $Z_0 = MonMult(X, R^2 mod M)$
2. For i=0 to (n-1)

   $P_{temp} = MonMult(P_i, Z_i)$

   $Z_{(i+1)} = MonMult(Z_i, Z_i)$

   If $e_i = 1$ then $P_{(i+1)} = P_{temp}$ else $P_{(i+1)} = P_i$

   End for
3. P=MonMult(P_n,1)

This is an overview of the algorithm implemented. The full algorithm is explained in detail in [HSS00].

After achieving functional encryption, we needed to optimize our Verilog to ensure the synthesized code would fit onto the SmartFusion. This was accomplished by using a dual-port RAM module to store the RSA modulus, exponent, message, and residue value, which reduced the number of registers needed in the fabric.

Furthermore, the addition and shift of three 64-bit numbers that occurs inside the Montgomery module (i.e. (A + B + C) >> 1) resulted in a synthesis that was still too large. By designing a state machine that ensured that only a single 64-bit adder would be needed on any one clock cycle, it was finally obtained the RSA module to synthesize down to a size that would fit onto the SmartFusion fabric.

Chapter 4

Software development

Chapter 4 describes all the software developed in order to interface the components and integrate all the parts into a solid, robust, coordinated and reliable system. This chapter follows a high-to-low-level structure that goes from the high-level description to the libraries developed to interface the devices as well as their drivers' development.

The first section explains the workflow of the state machine and its operation algorithm to control all the components and the intelligent door lock itself. This program describes the behavior of all the components integrated and the management of actuators output and values read by sensors.

The second section of the chapter refers to libraries developed in order to interface the components. Here it is possible to find more details about how the sensors work: the variables they read, the information they process, transmit, and receive. The libraries of the four main components (NFC sensor, LoRa transmitter, servo and NeoPixel LED) will be analyzed, specially the NFC drivers and LoRa connection implementation.

Finally, there is a section dedicated to the web server developed to store the information of the white and black list users in order to grant or deny access to owners, guests and strangers.

4.1 Main program

The main.c has three main functions used within its definition. The first two functions that are called, MSS_GPIO_init() and init_modules(), which initialize the I/O pins and modules connected to the SmartFusion, respectively. After those calls, TEST_state_machine() is executed, which is the function that controls the lock of the door depending on the inputs (server requests, transmissions and RFID

detections). This function is basically constituted by a while loop to check constantly the environment (NFC for instance), followed by multiple checks on the elements. The first check: to verify the state of the door. If it is opened, the color of the NeoPixel LED is set to yellow. In case it is closed, the system checks first if a response from the LoRa was received. If so, it decrypts it and reads it (it may have received a "grant access" or "block access"), otherwise the workflow leads to the NFC status: it verifies if a NFC tag is detected and it provides the identification to the SmartFusion, which sends a request to the server in order to determine whether the ID is on the whitelist or blacklist. Depending on the LoRa server response, the SmartFusion board will lock or unlock the door. Figure 4.1 shows the workflow diagram of this operation algorithm.

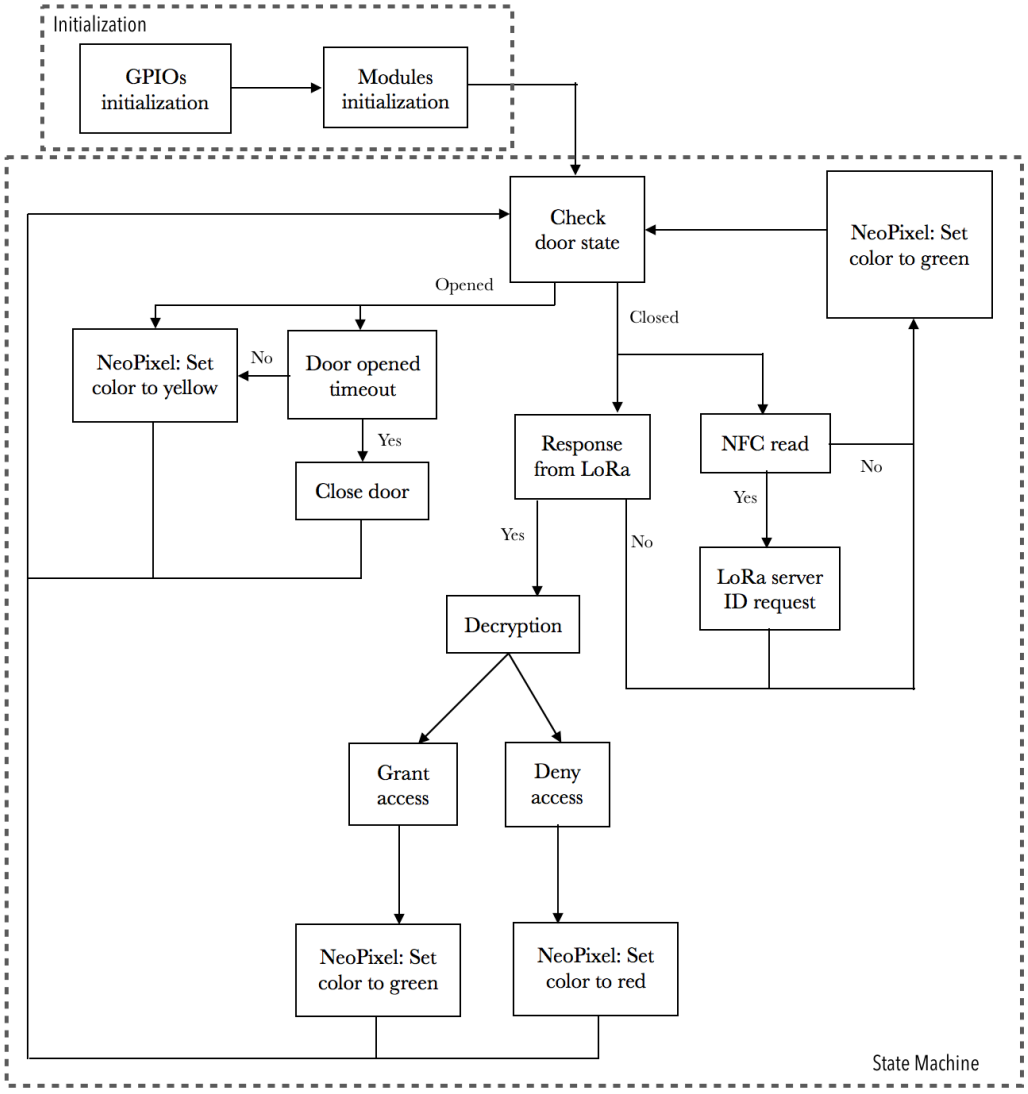


Figure 4.1: State machine diagram

4.2 Libraries

This section goes over every component and analyzes the implementation and development of each module. NFC drivers and LoRa communications between transmitter and hub are explained more in detail since they are the main parts in which the author participated.

4.2.1 NFC

The NFC PN532 sensor is a complex device that includes lots of functions and features applicable to any kind of project. In order to develop a full set of drivers it was required to use the datasheet provided by NXP, the most complete one found on the web (the PN532 that was purchased was manufactured by KNACRO, but the documentation provided had little information). This 200-pages datasheet provides any kind of detail regarding to the NFC functioning.

Drivers: data exchange

The PN532 NFC sensor offers three serial possible interfaces: UART, SPI and IC2. Nevertheless, SPI and IC2 are the most documented interfacing options, and in this project I2C is used for the communication with the controller.

The main task of the drivers is to communicate successfully with the device, so the microcontroller can send and receive necessary information. In this case, the SmartFusion will be using the advanced IC2 communication of the NFC sensor, which basically consists in a handshake mechanism combination that involves the triggering of interrupt (the P70_IRQ pin) that will let the processor (also called host controller) know when the information bytes frames are available to be read. Figure 4.2 shows a diagram in which a successive exchange of information takes place. The illustration shows a communication initiated by the host controller. This can useful to check the general status of the NFC sensor to check the device receives correctly the data and sends back the information requested (e.g. GetFirmwareVersion, a command that returns the current firmware version of the PN532).

The information exchange is not only important because it allows us to write on the sensor (like setting configurations) but also it allows us send back information, like the identification of NFC tags, the scope of the NFC interface.

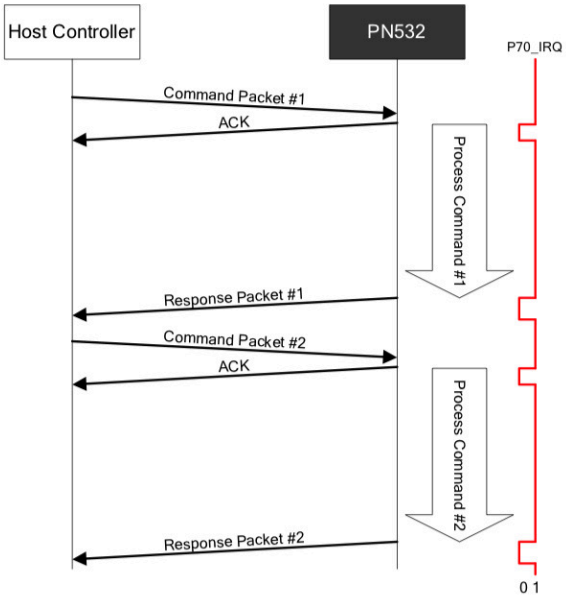


Figure 4.2: Information exchange between the NFC sensor and controller

As can seen from previous illustration, certain information is requested to the sensor in a command packet. The PN532 reports an ACK (acknowledge) frame to let the SmartFusion know it has received the information correctly and consequently, it processes the request and sends back the response packet when available. The host controller will perform reads until it receives response from the device (when the interrupt triggers). Something that should be pointed out is the fact that the Cortex-M3 interrupt handling does not permit the IC2 communications due to its buses use and interrupt priorities, i.e. the interrupt handler does not support IC2 reads and writes within the interrupt handler (see Cortex datasheet). This issue was found out during the drivers testing and was solved by setting flags to indicate whether or not it is possible to read the I2C buffer.

Drivers: command packets

Command packets constitute the way the SmartFusion sends and receives data from the sensor. The bytes sent using I2C interface follow a structured frame of bytes: preamble, start code, len, lcs, tfi, data, dcs and postamble, as it is shown in Figure 4.3. The information in detail about the frame and frame bytes can be found in the NFC datasheet.

Preamble: constituted by one byte (0x00).

Start code: constituted by two bytes (0x00 and 0xFF).

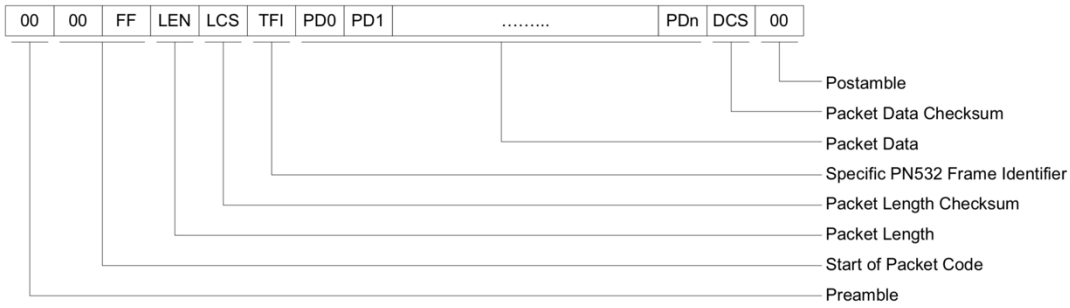


Figure 4.3: Normal information frame

Len: 1 byte indicating the number of bytes in the data field.

Lcs: 1 Packet Length Checksum LCS byte that satisfies the relation. Lower byte of [LEN + LCS] = 0x00.

Tfi: 1 byte frame identifier, the value of this byte depends on the way of the message. D4h in case of a frame from the host controller to the PN532, D5h in case of a frame from the PN532 to the host controller.

Data: length 1 bytes of Packet Data Information The first byte PD0 is the Command Code.

Dcs: 1 Data Checksum DCS byte that satisfies the relation: Lower byte of [TFI + PD0 + PD1 + ... + PDn + DCS] = 0x00.

Postamble: constituted by one byte (0x00).

All these bytes are wrapped into an array and sent/received through IC2 serial communications buffer. The function nfc_send_command() takes all these bytes (including the data) and their positions in the buffer into account and sends them through the I2C port using the built-in I2C MSS built-in drivers. In the transmissions of command packets is essential the management of the ACK and NACK packets exchange.

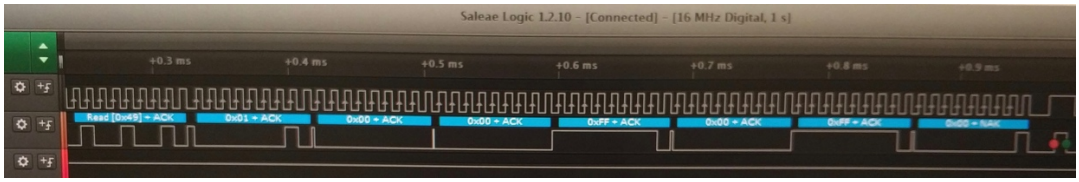


Figure 4.4: ACK frame decoded with Saleae analyzer

These types of frame are used for the synchronization of the packets. The ACK (Figure 4.4 shows an ACK packet exchange using Saleae logic analyzer[4]) frame may be used either from the host controller to the PN532 or from the PN532 to the host controller to indicate that the previous frame has been successfully received. In case of NACK (Figure 4.5), it is used only from the host controller to the PN532 to indicate that the previous response frame has not been successfully received, then asking for the retransmission of the last response frame from the PN532 to the host controller.

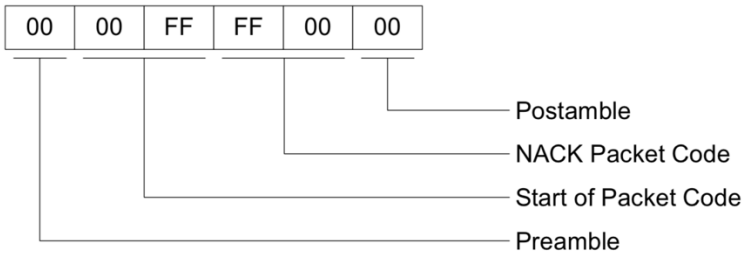


Figure 4.5: NACK frame

This means that the processor will be reading the buffer until it gets an ACK frame and it is considered in the library code development to interface and make functional the NFC sensor.

High-level code

The PN532 offers a wide offer of commands and functionalities, but for its application in this project just certain features/commands are needed. Commands such as GetStatus, GetFirmware or RFConfig were useful for debugging. The actual important packet is InPassiveListTarget, a packet of information that lets the host controller know how many targets are detected, their NFC IDs, and their length. However, prior this read transfer, the SmartFusion should indicate what kind of card, or in what mode is it going to be working. All this configuration is set using the SamConfiguration packet. Since the system only requires reading ordinary NFC tags, it is set in normal mode (first byte of the data frame of the SamConfiguration command is equal to 0x01). Also, the SamConfiguration sets the timeout of the read request, so this exchange takes places in the initialization of the NFC module. The initialization does not receive or return any value. It just sets digital signals for certain amount of time to initialize the device, executes the mentioned

[4] Software tool to record, measure, visualize, and decode signals in electrical circuits.

SamConfiguration and initializes the interrupt. Once the module is initiated, it is possible to use the InListPassiveTarget, the essential function to read RF fields and identifications. Once a RFID is detected, its NFC ID is stored on the NFC response buffer. This data is sent through LoRa transmitters to the webserver application, once the data from the command received has been selected.

4.2.2 LoRa

The software development of LoRa technology was carried out in two phases. The first priority was to create the drivers and be able to access to desired data, as well as ensure a communication with another LoRa transmitter. Secondly, the gateway should be set up so it can connect with the LoRa transmitter. Part of the information shown here has been extracted from the LoRa datasheet.

Drivers

There are two main sources from which the information of LoRa interface was extracted. In the first place, the Semtech LoRa datasheet: a 129 pages document where is possible to find every kind of detail regarding to the interfacing, memory managing, modes of operation, and all the electrical and digital characteristics. However, the documentation provided by Radio Head [Rld] was more useful since there is a C++ library file (RH_RF95) with drivers to send and receive unaddressed, unreliable datagrams via a LoRa capable radio transceiver. Great part of the work was converting the drivers to SmartFusion, while learning how control and data managing works inside the LoRa transceiver.

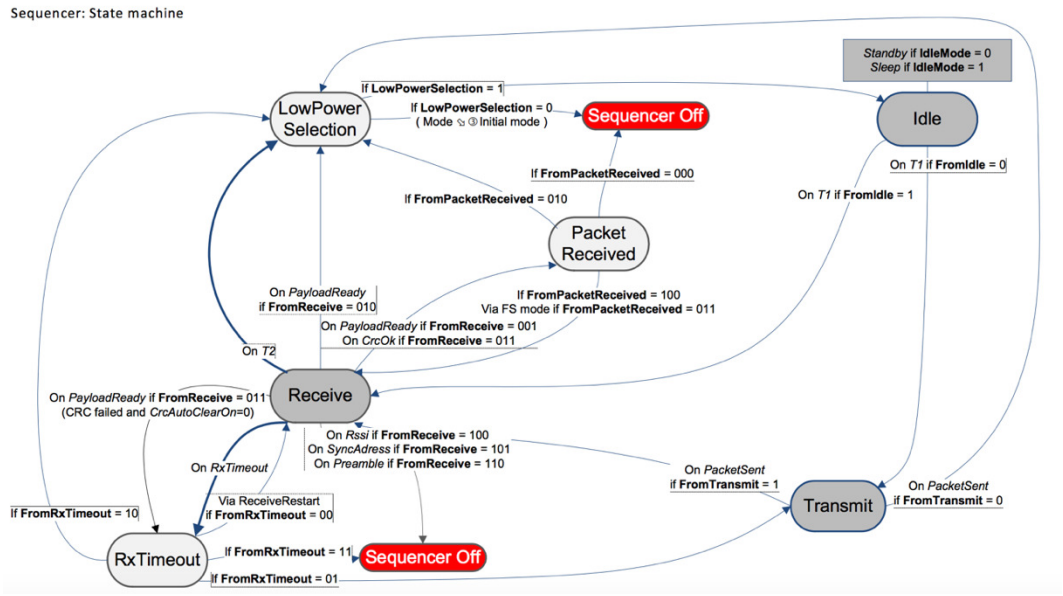


Figure 4.6: Sequencer state machine

This device has complex operation sequences, as can be seen in Figure 4.6. The transceiver can run into low power selection state as well as receive, packet received, rx timeout, transmit packet and idle. However, for drivers' development it is important to know how the flow of data works: where the relevant information is accessible and how the data path controls are managed. There are two operation modes: continuous mode (in which each bit transmitted or received is accessed in real time at the DIO2/DATA pin) and packet mode (user only provides/retrieves payload bytes to/from the FIFO stack. The packet is automatically built with preamble and word synchronization).

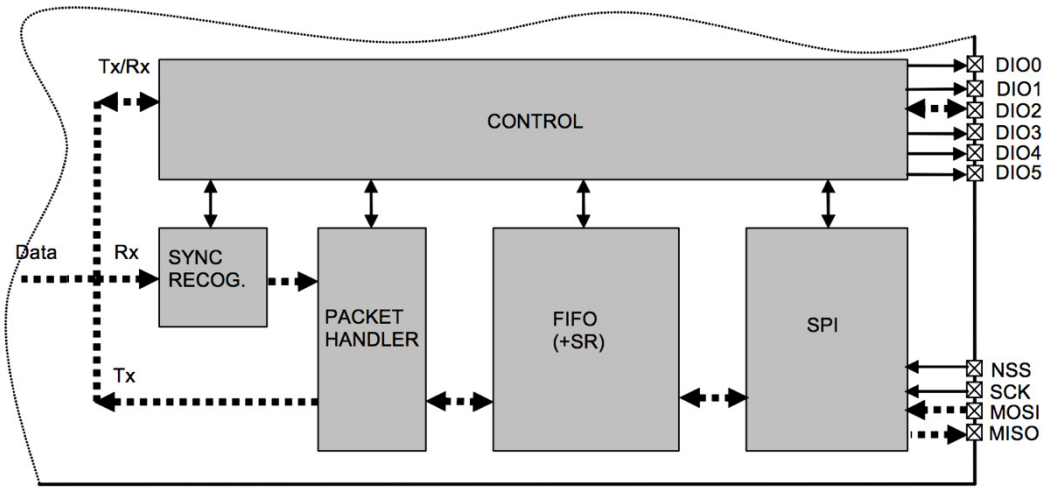


Figure 4.7: Data processing conceptual view

Once the data is received/transmitted by the LoRa module, it goes through three phases before it can be read/write through SPI interphase (Figure 4.7). The packet handler organizes the bytes to send and converts the information into packets with certain length. In packet mode of operation both data to be transmitted and that has been received are stored in a configurable FIFO shift register (Figure 4.8) (first information in, first out). It is accessed via the SPI interface and provides several interrupts for transfer management. The interrupts used for the shift register are described below:

- FifoEmpty: FifoEmpty interrupt source is high when byte 0, i.e. whole FIFO, is empty. Otherwise it is low. FifoEmpty state must be checked after each read operation for a decision on the next one.

- FifoFull: FifoFull interrupt source is high when the last FIFO byte, i.e. the whole FIFO, is full. Otherwise it is low.

- FifoOverrunFlag: FifoOverrunFlag is set when a new byte is written by the user or the SR while the FIFO is already full. Data is lost and the flag should be cleared by writing a 1 (as well as FIFO register).
- PacketSent: PacketSent interrupt source goes high when the SR's last bit has been sent.
- FifoLevel: Threshold can be programmed and the interrupt will trigger if and only if the maximum number of byte specified in the configuration is reached. Therefore, it is useful to indicate whether or not the register has the number of bytes needed to initiate communication (both read and write/transmit).

FifoLevel interrupt is used in the source code library in order to let the SmartFusion know when the data received from the server is ready to be read.

Connection test

Once the methods and attwere adapted, it was necessary to test if the SmartFusion was able to communicate through LoRa with other controller. To test the functioning of the LoRa communication on the SmartFusion side, an Arduino was used on the other side of transmission, which had previously been previously tested with another Arduino (with drivers and examples provided by Radio head):

```
#include "lora.h"
#include "drivers/mss_uart/mss_uart.h"
void LORA_client_to_gateway_setup (void){
        if (LORA_init() == 1){
                printf("init failed\r\n");
                return;
        }

        while (1){
                LORA_client_to_gateway_loop();
                printf("Test\n\r");
        }

}

void LORA_client_to_gateway_loop(void){
  printf("Sending to LoRa Server\r\n");
  uint8_t data[] = "Hello, this is device 2";
  LORA_send(data, sizeof(data));

  LORA_wait_packet_sent(0);
  // Now wait for a reply
  uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);
```

```
  if (LORA_wait_available_timeout(650))
  {
    // Should be a reply message for us now
    if (LORA_recv(buf, &len))
    {
      printf("got reply: ");
      printf((char*)buf);
      printf("\r\n");
    }
    else
    {
      printf("recv failed");
    }
  }
  else
  {
    printf("No reply, is rf95_server running?\r\n");
  }
  //delay(400);
  int i;
  for (i = 0; i < 10000; ++i);
  return;
}
```

After some tests and changes, the LORA_wait_available_timeout(), LORA_recv(), LORA_wait_packet_sent(), and LORA_send() functions worked accordingly as expected, both microcontrollers display the message that were sent. The test was successfully attempted, and the transceiver was ready to use.

Gateway set-up

Dragino offers a convenient way to set up the gateway by downloading and configuring the Arduino IDE. After adding the Dragino boards available in their repository, it is possible to access to the gateway by adding the Arduino Yun port (gateway and computer should be connected to the same network to work). After the configuring process, a new test was performed with the gateway similarly as it was done with LoRa transceiver-transceiver communication. This test only requires to install the Radio Head LoRa library in the Arduino IDE and upload sketch to LoRa client[5]. Running the script will provide a monitor for debugging that will display traces to ensure there exists effective connection between the transmitter and the hub. Next step was to implement the webserver in the dragino.

[5] Procedure explained in detail in Dragino gateway datasheet

4.2.3 Servo

The servo motor is the component in charge of locking the door by fixing its angular position to close or to open. This device is controlled by voltage: the more power transmitted, the longer angular position it can reach. Therefore, this device is controlled by a PWM that has a fixed period and a variable duty cycle. The servo's angular position should vary from 0º to 90º to lock the door and vice versa to unlock it. The implementation is mainly done in hardware, using the APB3 bus interface to increase or decrease the duty cycle. The bus address is set to 0x40050004 (according to the ARM v7-M Architecture Reference Manual addresses 0x40000000- 0x5FFFFFFF are reserved for peripherals). The values of the adjustable duty cycle are set in software by using the functions SERVO_lock(), SERVO_unlock(). SERVO_read_state() provides the state (duty cycle previously set) of the lock. After setting the values, they are applied in the fabric using the servo hardware module via MMIO.

4.2.4 NeoPixel LED

NeoPixel LEDs use their own serial protocol interface. The drivers can essentially apply three actions to the LED: set a colour (NP_set_pixel_c()), apply a color (NP_apply(), does not change colour) and clear the colour (NP_clear()). The SmartFusion can apply these functions with color green, yellow, red and blue, with codifications 0x00901030, 0x0083A400, 0x00209925 and 0x00200088 respectively.

4.3 Web server development

The Arduino side code was written for sending and receiving data as well as properly converting that data into usable forms. However, the gateway was useful due to its Linux environment and the infinity possibilities that offers the Arduino bridge connection library. The bridge Library use UART port to communicate between MCU and the Dragino AR9331. Below is the block diagram shows the bridge connection between the Mega328P MCU and Linux.

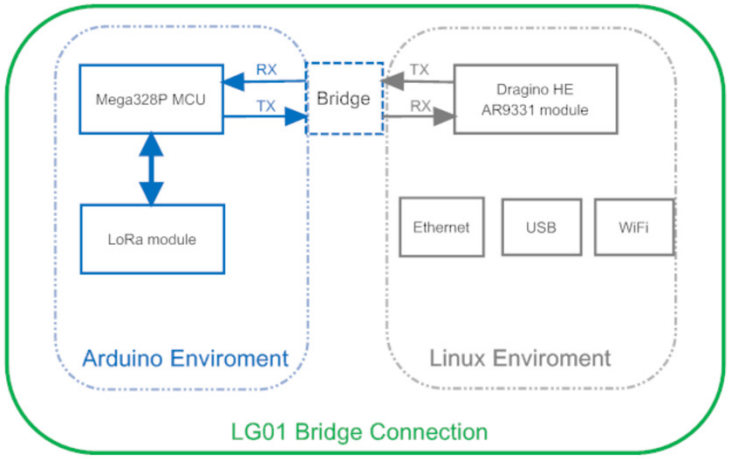


Figure 4.8: Dragino gateway bridge connection.

Linux side code was written in Python as a Bottle server (firstly it was tried to use Flask but there was not sufficient memory on the Linux environment to use it). The server is characterized by a minimal interface and account-based access system. Figure 4.9 shows a screenshot of the sign-in page and Figure 4.10 the main page of the server application.

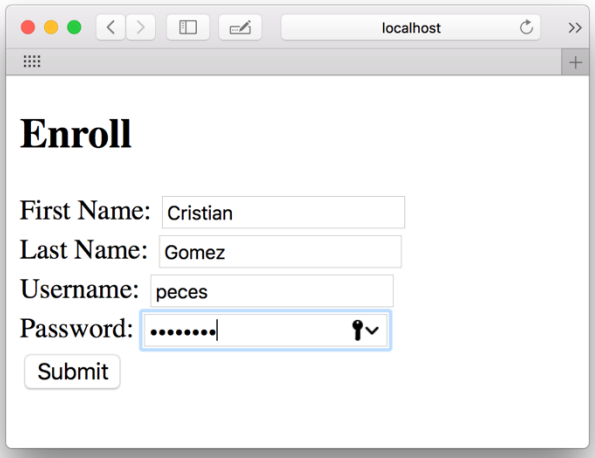


Figure 4.9: Enroll webpage.

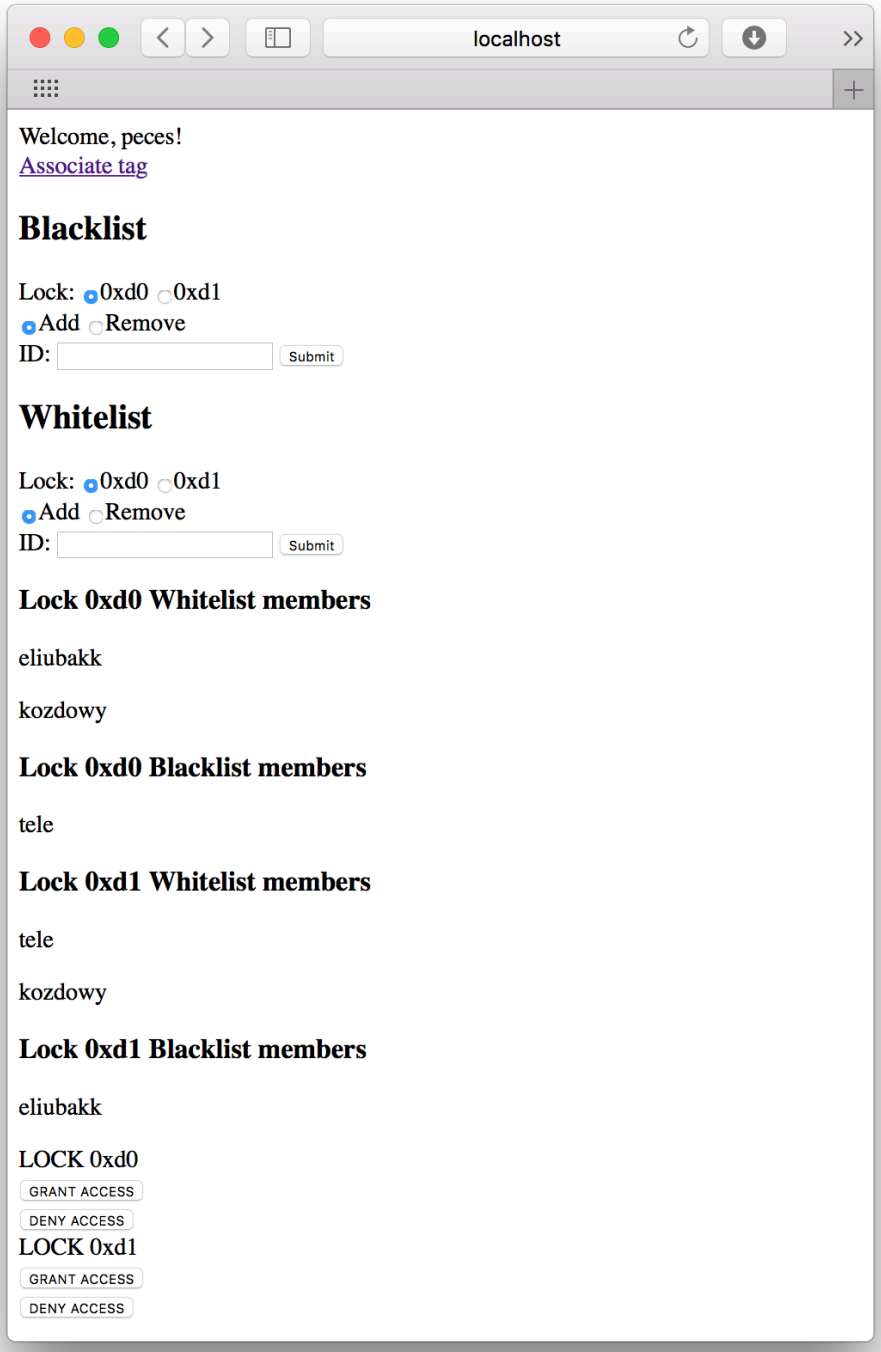


Figure 4.10: Main webpage of the server application.

Chapter 5

Results

This chapter wraps up all the development that was carried out along the first part of the project. The achievements will be compared and contrasted with the initial objectives and approaches that could be applied to enhance the operability of the system. Finally, the last section goes over further work that could be implemented in order to make the system more complete, reliable or simply to add extra features.

5.1 Conclusions

The section of conclusions is divided into two parts. In the first place, there is a contrast between the goals that were supposed to be achieved and the final results of the exposition. Second part explains which of the elements in the system could be improved in order to make the device better system.

5.1.1 Goals accomplished

All the stated objectives were achieved successfully, although the integration of some modules was performed more satisfactorily than others. These details are analyzed below.

The interface of the NFC sensor, LoRa transceiver, servo, and NeoPixel LED was a total success. The devices worked accordingly as they were expected to work in the specifications. The PN532 was able to read NFC tags and some smartphones configured for that purpose. The SmartFusion was perfectly able to communicate with the server application through LoRa technology. The servo was able to turn 90 degrees accurately and the NeoPixel changed its color according to the door state with high reliability.

The server application worked perfectly integrated within the gateway and the SmartFusion. It was able to store whitelist and blacklist IDs, decrypt data and also

provide a user interface in order to interact with the owner(s) of the house. However, the server user interface was minimal and would not be ready for a commercial use since it does not have a good-looking user experience. The server was accessible from the private network and an account-based system was implemented, i.e. it included functionalities such as granting remote access or list-checking.

The RSA encryption/decryption worked perfectly fine and carried out its task with no failure. The main downside is the lack of sufficient space to allocate all the modules necessary for the hardware implementation, so after a intense work of optimization it was only possible to design a state machine that handles a 64-bit encryption and no protocol was designed to combat replay attacks. The system could be unsafe in case of an attack. However, is works as a proof of concept.

On the other hand, it was not possible to achieve the implementation of the fingerprint sensor (which was set as a secondary objective), because of the lack of time. The fingerprint sensor would have provided an extra item-less entry method, as well as a method that didn't have to connect home to the server since the sensor maintained an internal list of fingerprints. During the initial part of drivers' development, it was realized that the sensor that was bought didn't have any sort of interrupt pin and would need an external sensor like a capacitive touch sensor to let the SmartFusion know a person was nearby; otherwise the sensor would need to be continuously polled, wasting lots of clock time and power.

The door and door frame design for the exposition were designed using SolidWorks. The design plans can be found in the annex of the document. Figure 5.1 and 5.2 shows an image of the rendered objects.

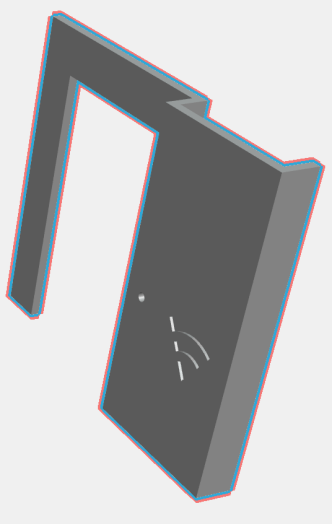


Figure 5.1: Door frame

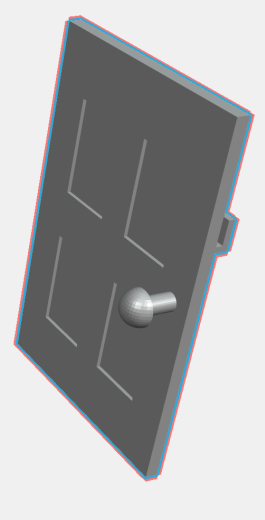


Figure 5.2: Door

5.1.2 Elements to improve

Some of the components that were mentioned in the previous section can definitely be improved to enhance system's operability.

In terms of the account-based system created in the webserver application, the minimal account system could be transformed into a democratized system, in which not just one owner of the house agrees to accept a guest but all of them. Ideally, it should incorporate an approval process for adding or removing IDs from the whitelists and blacklists, as well as a system to notify the users about remote access request.

As was mentioned, the connection to the server was encrypted but vulnerable to replay attacks, so it would be necessary to implement the system into another board. Another approach could be the use a more powerful FPGA (in terms of memory to allocate sufficient hardware modules to keep the system safe). Additionally, the communications should integrate some sort of protocol in order to combat replay attacks

5.2 Future development

Future development gathers the ideas that due to lack of time were unreachable to get. The integration of these features could contribute to the aim of the project. There are two main features that would be necessary and interesting to implement in the system.

The SmartFusion is meant to function under ideal work conditions, i.e. the system is installed to work within a stable environment, in which ambient variables do not change radically. Nevertheless, it would be reckless to not think about cases in which an intruder tries to break the module not just digitally but also physically. In other words, a person could use different approaches to disable the integration of the system. There exists essentially two ways of disabling an electronic system that is connected to the door: by disconnecting the general power supply or by applying a strong electrical shock through the contact with a high-power device. In order to solve the power supply dependency, it would be enough to connect a external battery ready to use in case of an emergency, and a notification algorithm would be convenient to let the owners know that a change in the power supply has been produced. In case of electrical shock, it would be required to attach an isolating device that acts as a Faraday cage in order to avoid any damage that could cause to the device. That is, we should add external electrical protections.

On the other hand, even though the device aimed to be an item-less system, it ended up being a key-less but not item-less system, since the use of a NFC tag or smartphone is still necessary to make it work. Therefore, it would be convenient to implement some sort of biometrics sensor to achieve this objective. Today, the easiest biometrics sensor to implement is the optical fingerprint reader, but other approaches can used such as face recognition. In fact, the goal of part II is to obtain an item-less authentication system by integrating a facial recognition module to the SmartFusion.

5.3 Exposition

The exposition of the project took place on 11th December 2017 at the EECS atrium in the University of Michigan. All groups from EECS 373 had to present a public exposition of the product not only to professors but also to the curious people that walked across the hall. Figure 5.3 shows the SmartFusion connected to the different sensor and devices at the back of the door. The names of the devices are also shown.

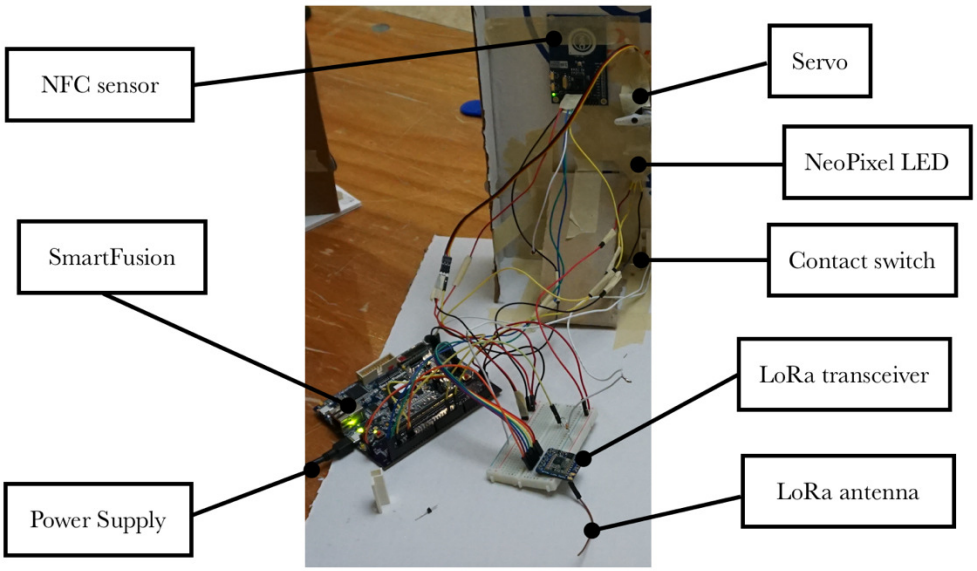


Figure 5.3: Prototype for final exposition.

Part II

Facial recognition

Chapter 6

Introduction to face recognition

Part II of this project was a University of Michigan EECS course: 499 Advanced Directed Study. This independent study was supervised by Matthew Smith, director of the project.

Although last part was a complete project, it was lacking some aspects that it was trying to accomplish. One of these elements is the absence of a system that needs no physical instrument to unlock the door (the current development stage of the project requires the usage of a cellphone or NFC tag to open the door). This aspect will be fixed with the incorporation of a face recognition module that will be running on a Raspberry Pi using the open source computer vision library OpenCV. This library has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, and Mac OS, with a supportive community that makes it suitable for its use in this project.

There are also other approaches that could be used to achieve a secure entrance without any device, using other parts of the human anatomy such as fingerprints or eyes iris detection that could be implemented by adding more sensors to the SmartFusion module, but after taking into account all the implications, it was considered that learning Python or familiarize with OpenCV software would be a much more enriching experience with challenging objectives. In this part, new concepts are explored, like computer vision and machine learning applications that OpenCV provides as well as the potential of python to use this library in particular, its capacity of interfacing devices, interacting with the operating system files or opening serial ports for data transfer. Since the Raspberry Pi runs a Linux distribution (Raspbian OS), it is necessary to know the basics of this OS and bash language fundamentals, for example for SSH tunneling as well as fundamental files management in the Raspberry Pi.

There are many elements to learn with regard to this part of the project. Thus, the learning curve will be larger than in previous part and therefore more time to

learn the tools will be needed since most of the concepts and tools in part one had been used along the term before the beginning of the project. This fact is reflected later on the methodology section, where one can see that the first three weeks of development were dedicated to do research and get familiarized with the tools that have been used.

6.1 System operation

In this section, it is introduced an overview explaining how the module should work, the global objectives and the specific tools that are used to achieve a successful integration of the whole subsystem.

6.1.1 Overview

The Raspberry Pi operates as an independent subsystem that communicates with the SmartFusion only if it is verified that the face of the person in front of the door belongs to any of the users previously loaded into the internal database. Therefore, the first step before initiating the face recognition algorithm is to check that someone is actually there, so the program does not need to be run all time, but only when a body is detected. This approach improves efficiency since face recognition uses significant part from the processing resources, and it is done using a PIR sensor. The PIR sensor lets the Raspberry know if a human body has moved in or out its range, so the face recognition process can begin identifying faces, using a HD camera that is connected via USB. After several iterations, the algorithm determines whether or not the person is in the database, encrypts the unlocking/locking message and sends it through a UART connection to the SmartFusion, which will decrypt and process it to open the door.

6.1.2 Tools

Since the Raspberry is connected with the SmartFusion, Libero and SoftConsole will be used again to interface the new module and to process the data it sends.

Python is fundamental since all the code that is written in the Raspberry Pi uses Python. For this application, Python as a programing language makes the files processing, data management, time control and serial port usage much simpler. In other words, Python is very useful not only for its simple syntax but also because of all the libraries that are implemented and ready to use. In particular, OpenCV is

not installed by default but it can be easily installed following the instructions set in opencv.org or using a package manager.

"OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms" [Ogi]. Face recognition is one of its multiple machine learning applications. "These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies" [Ogi].

6.2 Purpose

This second part of the project aims to develop a contactless system that requires no device to unlock the door by using an algorithm capable of identifying someone's face and correlate it with a given database. In this section it will be discussed the objectives involved in the development o and the considerations taken into account to achieve those objectives.

6.2.1 Objectives

There are three main goals in part II:

- **Develop an effective and reliable face recognition application.** This objective involves writing the code of the three main sections of the face recognition process, adapting them to the OpenCV version installed in the Raspberry Pi and run several tests to determine what is the appropriate correlation to set between the trained models and the actual face in order to output a positive response to the SmartFusion.

- **Connect the module with the SmartFusion properly.** The Raspberry Pi and SmartFusion should communicate with each other somehow. It should be a reliable connection (some kind of encryption could be added as an additional secure factor) that can transmit data processed in python to C in the locking module.

- **Final integration.** All the submodules developed in the new system should be working appropriately in the Raspberry side. In addition to this, the final integration with the SmartFusion should be effective to ensure a correct functioning of the locking system.

6.2.2 Considerations

There are several approaches to achieve those objectives, and this is the main reason why several milestones took place along the development period. For instance, there are multiple manners of doing the connection between the Raspberry and the SmartFusion.

LoRa transmitters and Bluetooth devices were considered to carry out this task. LoRa was a good option since the locking system is based on this technology and it would not be difficult to create a new library that process the data sent from a LoRa transmitter from the Raspberry; in addition, there exists Python libraries made for the Raspberry Pi that interfaces easily the LoRa transceiver. Its downside: it would increase the cost of the potential final product. On the other hand, there are Bluetooth cheap wireless receivers and transmitters and relatively easy to implement, but as it was showed in chapter 1, one of their main lack is the absence of a secure wireless connection, what makes it unsuitable for this application. Instead, this issue is solved using a simple UART connection: straightforward to implement and secure since it is based on wired connections and the odds to get jammed are very low compared with the other approaches.

The way human presence should be detected was discussed in the first general milestone at the end of January (see methodology section). The first approach considered was the use of an ultrasonic sensor (such as HC-SR04) that could detect if someone was approaching to the door. The face recognition process would start when the person is close enough. This configuration would save energy since the face detection code would not be running the whole time, although it would still be consuming a little portion of power to do constantly calculations to determine if the body is close enough. PIR sensor would solve this problem due to its simplicity of functioning. PIR sensors consume minimal quantity of power and can tell the

microprocessor whether or not there is someone in front of the door (it does not provide range between the sensor and the human body as the ultrasonic sensor would do, but such additional data is not required for the purpose of its application), which is all what is needed for this part of the system. The convenience of using PIR sensor makes it the most suitable approach to detect human presence.

6.3 Working methodology

The working methodology consisted in the achievement of objectives and suitable adjustments as time progressed. Once or twice a month took place milestones in order to analyze the progress of the project and discuss different approaches workable to get working some parts of the system. In figure 6.1 one can find the Gantt diagram, where it is possible to see the dates established for project management. In total, there are 9 different tasks and 3 general milestones distributed over the 12 working weeks' time, which can be summed up in the following tasks bellow:

1) Week 1: Installation of Ubuntu and needed drivers: Wi-Fi, iSight, etc.
2) Weeks 2 - 4: Project documentation
3) Week 5 - 6: Implementation of multiple faces recognition in computer
4) Week 7: Adaptation for Raspberry
5) Week 8: Spring break
6) Week 9: Midterms week
7) Week 9 - 10: Complementary modules development: timer, PIR, email library, UART communication, Python XTEA encryption implementation and SmartFusion library creation for communication and encryption.
8) Week 11: Final integration
9) Last 8 weeks: Write-Up

The Raspberry Pi can be set up in different modes depending on what was the purpose of the task to test/implement. For example, for testing the PIR sensor, it is sufficient to set it up in headless mode: the Raspberry Pi is controlled remotely by stablishing a SSH tunnel in the same private network (so no monitor or keyboard is needed to test the sensor). However, sometimes it is necessary to see what is displayed in desktop. For example, to verify the algorithm is functioning correctly,

in particular to check if the faces were recognized. In these cases, there are two main approaches that can be used. The first option consists in connecting the little computer to a monitor using a HDMI. The second approach consist in using the VNC server and viewer to connect the Raspberry to the personal computer and watch what is displayed remotely from the personal computer screen (desktop remote access). For this project, the first approach was used for convenience in lab.

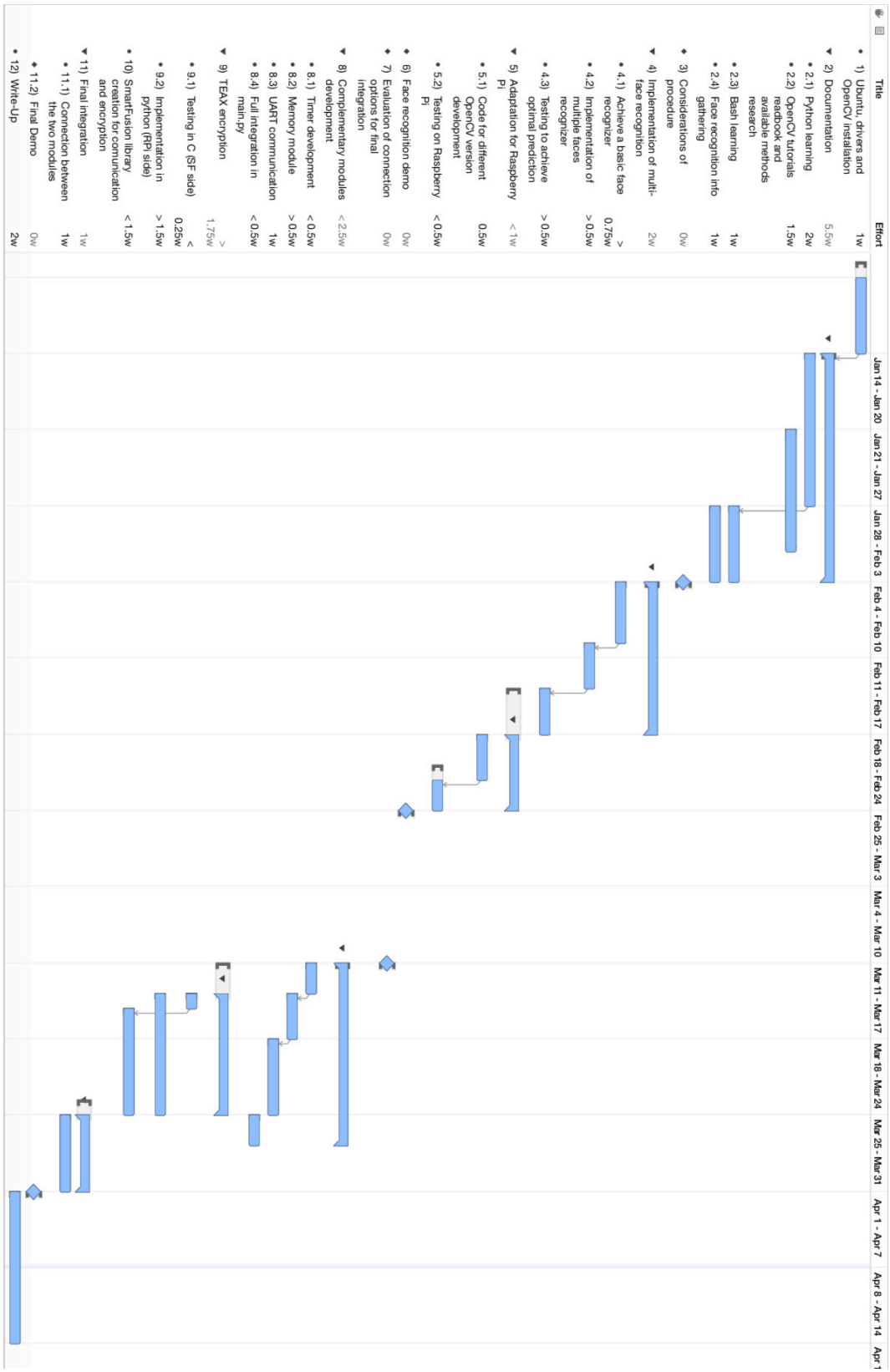


Figure 6.1: Project management planification

6.4 Elements

There are 3 main components used for the full integration of the face recognition module: Logitech HD Pro Webcam C920, PIR sensor and the Raspberry Pi.

6.4.1 Logitech HD Pro Webcam C920

This camera is used to capture the faces from people that stand in front of the door. It is connected through the USB port and provides high definition of the pictures, what that makes it suitable for this application since quality of facial features is highly recommended to develop a reliable face recognition system. It costs $50 in Amazon and its features include Full HD 1080p quality video, 15 MP photo quality, H.264 encoding[6], Full HD glass lens technology, 20-step autofocus and built-in microphone that will not be used for the purpose of face recognition. In figure 6.2 can be found a picture of how this camera looks like.

The last point to emphasize is the possibility of using it in a Linux distribution such as Raspbian OS without installing any additional driver. This makes it very convenient for the project.

Figure 6.2: Logitech HD Pro webcam C920

[6] H.264 or MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) is a block-oriented motion-compensation-based video compression standard.

6.4.2 PIR sensor

PIR sensors allow one to sense motion, and it is usually used to detect whether a human has moved in or out of the sensors range. They have some of the features that make them convenient: they are small, inexpensive, low-power, easy to use and do not wear out. For that reason, they are commonly found in appliances and gadgets used in homes or businesses. They are often referred to as PIR, "Passive Infrared", "Pyroelectric", or "IR motion" sensors. PIRs are basically made of a pyroelectric, which can detect levels of infrared radiation. Everything emits some low-level radiation, and the hotter something is, the more radiation is emitted. The sensor in a motion detector is actually split in two halves and it detects motion (change) not average IR levels. An image of the PIR sensor is showed in figure 6.3. The information above was extracted from PIR sensor datasheet. This sensor will be connected to the Raspberry Pi and located in the front of the door frame in order to detect the irradiation from incoming bodies.

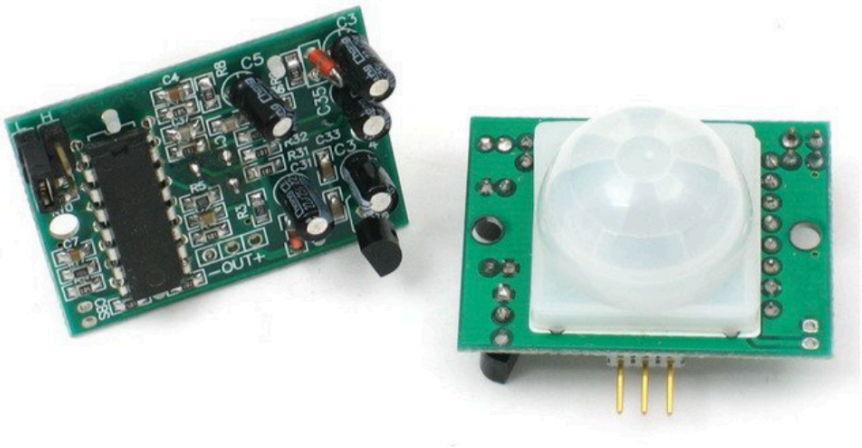


Figure 6.3: PIR sensor

6.4.3 Raspberry Pi

Raspberry Pi is a small, powerful, cheap, hackable and education-oriented computer board introduced in 2012. This credit card-sized computer with many performances and affordable for ca. $30 is perfect platform for interfacing with several devices [PK15]. The Raspberry Pi operates in the same way as a standard PC, requiring a keyboard for command entry, a display unit and a power supply. Flash memory card normally used in digital cameras is configured in such a way to work as a hard drive to Raspberry Pi's processor. The unit is powered via the micro USB connector. Internet connectivity may be via an Ethernet/LAN cable or via Wi-

Fi wireless connection [VM14]. All these connections can be seen in figure 6.4. This little computer uses a flavor of Linux as operating system: Raspbian OS. The main advantages of using Linux is that it keeps low its price and there is also a great amount of open source software that can be used for many different types of applications. In this case it is used the OpenCV library. Additionally, since it became a popular platform, there are lots of documentation not only about the Raspberry features itself, but also about open source software developed for Linux. The version that will be used for this application is RPi 3 Model B (newest generation), which has the following specifications (provided by manufacturer):

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

Figure 6.4: Raspberry Pi 3 Model B

Chapter 7

Software and Hardware Implementation

The Raspberry runs a Python main program that uses different modules developed to interface external devices, system files, serial port connections, encryption algorithms and obviously the face recognition algorithm. On the SmartFusion side, the serial connection should be configured as well as the decryption algorithm to receive and process the information. Finally , it unlocks or locks the door depending on the person face correlation with the preloaded model. In this chapter it will be analyzed all the modules that are included in the main.py file. This elements are necessary for the proper operation of the entire system. The different tradeoffs and approaches used to achieve a correct functioning of every part will be discussed, as well as the ideas considered to solve certain problem encountered while testing. These are the modules that will be analyzed in the following sections:

- Recognition: involves the development of all the functions need for face recognition.
- PIR: module that initializes and interfaces the PIR sensor to use it previous face recognition process.
- Memory and Database: module that permits the interaction with the OS files and storage data of the people registered.
- Timer: time is an essential variable in terms of energy saving optimization.
- Encryption: secures the data sent to the SmartFusion.
- Communications: opens and uses the serial port to send data
- Email: to send the owners alert emails in case there is a potential threat.

7.1 Recognition

In this section it is explained how the face recognition works using OpenCV: the different steps to get a model and use it in order to identify the face of users. Some parts of the algorithm are discussed to get a basic understanding of the code.

7.1.1 Classifiers

OpenCV includes a Cascade Classifiers which let us detect the main characteristics of a certain object: there can be face classifiers, eyes classifiers, plane classifiers, etc. OpenCV comes with a set of different classifiers that can be used with computer vision purposes. In this case, OpenCV includes in the opencv/data/haarcascades/ folder a frontal face cascade classifier that has everything needed for the face detection task. In case that the application needs to identify another type of object that has no classifier included, it would be required to create a new one and train it [Ofrd].

With the classifier it is possible to identify a face from a picture, and it will be used to take and store pictures that will be used by the model trainer as well as to display the video recording with the detected face around a rectangle (which is labeled with the corresponding name of the person).

7.1.2 Face recognition stages

There are three main different parts in the face recognition process [Eofrma]: data gathering, model training and detection. This process is displayed in figure 7.1:

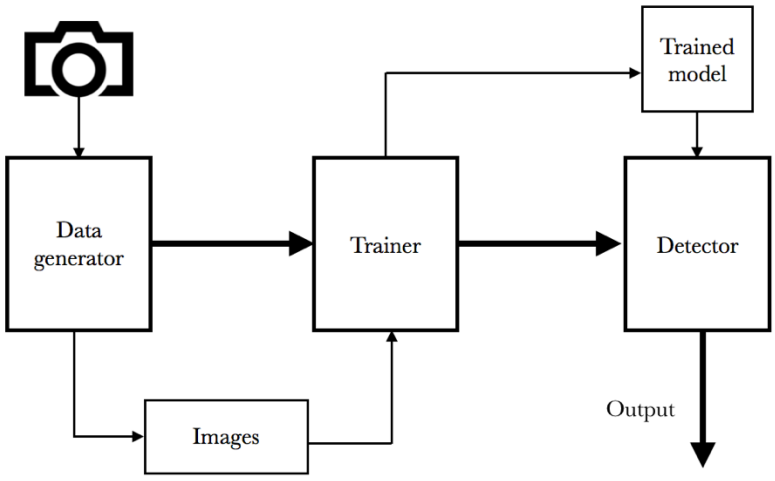


Figure 7.1: Stages of the face recognition process

Phases:

- **Data generator.** Multiple photos are taken from video recording. These pictures are resized and changed to gray color for later training process. Finally, they are sorted and stored in a folder called 'dataset' with a particular nomenclature: each picture has its own number and belongs to a particular person (id number).

- **Trainer.** The photos that were stored in the previous stage are loaded and, using the front face classifier, the pictures are labelled and processed by training process, which yields at the creation of a .yml file that constitutes the model that is used to recognize that person.

- **Detector.** The model trained is loaded to use it in a real time picture prediction. The face is detected – again using the front face classifier – and processed with the predictor of OpenCV. If the method obtains a high correlation, it is assumed that the person face captured is the same as the one stored in the data base.

7.1.3 Algorithm

OpenCV includes three different algorithms for face recognition: Eigenfaces, Fisherfaces and Local Binary Patterns Histograms. Eigenfaces and Fisherfaces find a mathematical description of the most dominant features of the training set as a whole. LBPH analyzes each face in the training set separately and independently. The LBPH method is somewhat simpler, in the sense that the images are characterized in the dataset locally, and when a new unknown image is provided, it is performed the same analysis on it and compared the result to each of the images in the dataset. The way which the images are analyzed is by characterizing the local patterns in each location in the image. The LBPH method works better in different environments and light conditions, however, it will depend on the training and testing data sets. It is needed around 10 different images of this person's face in order to be able to recognize him/her correctly ([SKP15],[Ofrd]). Since LBPH is a robust face recognition algorithm, it is used for the project and some of its fundamental will be explained below to have a basic knowledge of the way pictures are processed by the system.

The LBP operator was originally designed for texture description. The operator assigns a label to every pixel of an image by thresholding the 3x3-neighborhood of each pixel with the center pixel value and considering the result as a binary number. Then, the histogram of the labels can be used as a texture descriptor [AHP06]. See figure 7.2 for an illustration of the basic LBP operator.

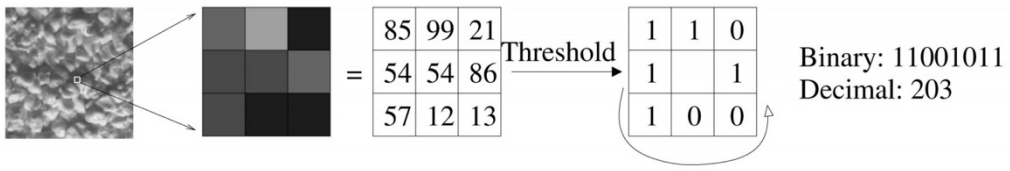


Figure 7.2: LBP operator [AHP06].

As it is showed, once the threshold is applied, the algorithm obtains a binary number according for each pixel. With 8 surrounding pixels there are 2^8 possible combinations, which are called Local Binary Patterns [Ofrd]. The pixel neighbors can also be determined in circular patterns, which are called circular neighborhoods. This information essential for the creation of histograms, that will be the base for face characterization.

The methodology of the algorithm can be summed up as follows: the facial image is divided in several facial regions with their own texture descriptors, which are extracted independently one from each other. These descriptors are evaluated and concatenated to obtain a global description of the face [CL13]. This approach permits the creation of a model that can predict if a certain face belongs to a person based on the spatial relations of facial regions.

Since by definition the LBP is a robust operator using gray scale transformation, it will be needed to convert to gray scale the pictures two times (see Figure 7.3); firstly, in the generation of data to process it and generate the trained model (each of the pictures that are taken are converted into gray scale). Secondly, once the model is loaded, it can do the prediction if the selected image is in gray scale as well. Hence, the image will be converted into gray scale before obtaining the prediction. The way this process is done will be explained in the following sections.

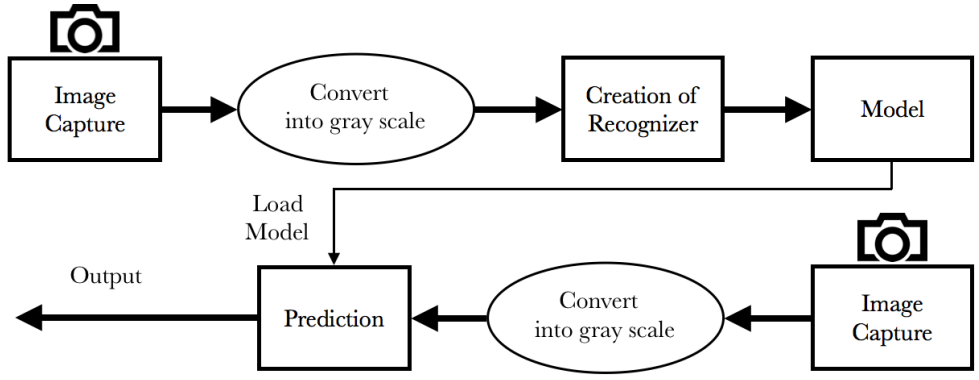


Figure 7.3: Gray scale transformations in the prediction process

7.1.4 Creation of recognizer

The process to create the recognizer used to determine if there exists a correlation between the face of person in front of camera and the face stored in the database involves the two first phases of the facial recognition process described above: data generation and data training. What is meant by data generation is the set of steps needed to obtain pictures of the subject face, apply some changes to the original picture and store them in a data set of photos. The data training stage involves loading all the pictures stored in the previous step and create a trained model based on the patterns and characteristics detected in the different images. The model is stored again so it can be used in the detector stage. The model associates given images with given IDs, so it is able to detect faces correlated with the used images and associate them with the ID given when it was trained (that is the function of the memory – database module).

Data generation

First thing to do is gather data about the new user. The first lines of code gather the user's name and generates an ID number based on the current number of users in the data base. In addition, the camera connected to the Raspberry Pi is opened as well as the frontal face classifier with the VideoCapture(0*)* and CascadeClassifier(<name_of_classifier>) methods included in the OpenCV library (import cv2).

After the initialization, the program starts a loop that captures the images and process it using read() and cvtColor(), which converts the image into gray scale in order to apply the face classifier (loaded as detector) and obtain the face

characteristics of the face showed on the photo. Immediately after detecting the special characteristic of the face, it is showed in the screen and saved into the file system with the imwrite() and imshow() methods, respectively. This process is repeated with every picture until the program reaches the upper limit bound of pictures to take (N_PHOTOS_TRAIN).

Once the images are saved, the window frame generated is closed and the camera is realeased with the destroyAllWindows() and release() methods respectively. The data generation is over and the next step is to train the face recognizer with all that information stored in pictures.

Data training

The first thing to do is to create a LBPH recognizer, which is made using the method createLBPHFaceRecognizer(). Depending on the OpenCV version used, this function might be found as createLBPHFaceRecognizer() or as LBPHFaceRecognizer_create() in versions of OpenCV 3.3 or higher. In addition, in the last releases, the method comes defined in a submodule of cv2 called face and thus we use cv2.face.LBPHFaceRecognizer_create() to create the recognizer. Since the version installed in the Raspberry Pi is the 2.2, it is enough by using cv2. createLBPHFaceRecognizer() and all the code used works properly in OpenCV 2.2 and Python 2.7.

Before training the data, the data saved is opened and processed using the handmade function getImagesAndLabels(). This function obtains the face characteristics and assigns them the corresponded ID, loaded from the files name. This function receives the path of the images and the classifier, and returns the set of face characteristics and another array of the IDs that corresponds to those faces. In the first place, the face and ID arrays are initialized then the program gets into a for loop that goes over all the gray scaled images stored in the indicated path. To process every of the pictures, it is used the pillow library, which incorporates methods to load the images in black and white using the 'L' mode [Ppld]. Once the data is loaded, we convert it into a numpy array (the classifier input should be an uint8 numpy array), and plug it into the detectMultiScale() method that provides the classifier. This function detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles, and each rectangle contains the detected object. Finally, the objects are appended to the initialized list of face characteristics as well as the ID numbers in the ID list. These two objects are return to the main function, where they will be used to train the recognizer using train(), a

method incorporated in the LBPH recognizer class. Finally, the model is created and stored in an independent folder under the .yml extension.

7.1.5 Face detection using the recognizer

The face detection is carried out by the Detector function in the recognition module. Firstly, the classifier is opened again and the recognizer (model created) is loaded by creating the LBPH recognizer and then loading the model using the load() method. The operation algorithm works as follows.

The detection will be working for the time specified in WAITING_TIME (30 seconds) and a timer is used to specify the timeout. If no one is detected after timeout, the function returns False. In each iteration, a picture is taken and transformed into gray scale, processed by the classifier (getting the rectangle that contains the detected objects) and analyzed in a python-for loop which takes the values of the objects contained in the rectangles and elaborates a prediction whether or not the face detected is in the database (included in the model). If so, the data base of names is loaded (using the memory module) and the name of the person recognized is displayed in the screen using show(). The prediction provides a correlation between the face detected and the trained data. This parameter will be adjusted optimally to get reliable detection without producing false recognitions (see results section for more information). Next, the algorithm assign True to the returning variable if and only if at least MIN_TIMES times consecutively the detector recognizes the face. Finally, after the loop is done, there is a final check: if the person was not detected, the last image that was taken is saved in order to eventually notify the owners by email who is trying to enter into the house. The algorithm of recognition operation can be found in figure 7.4.

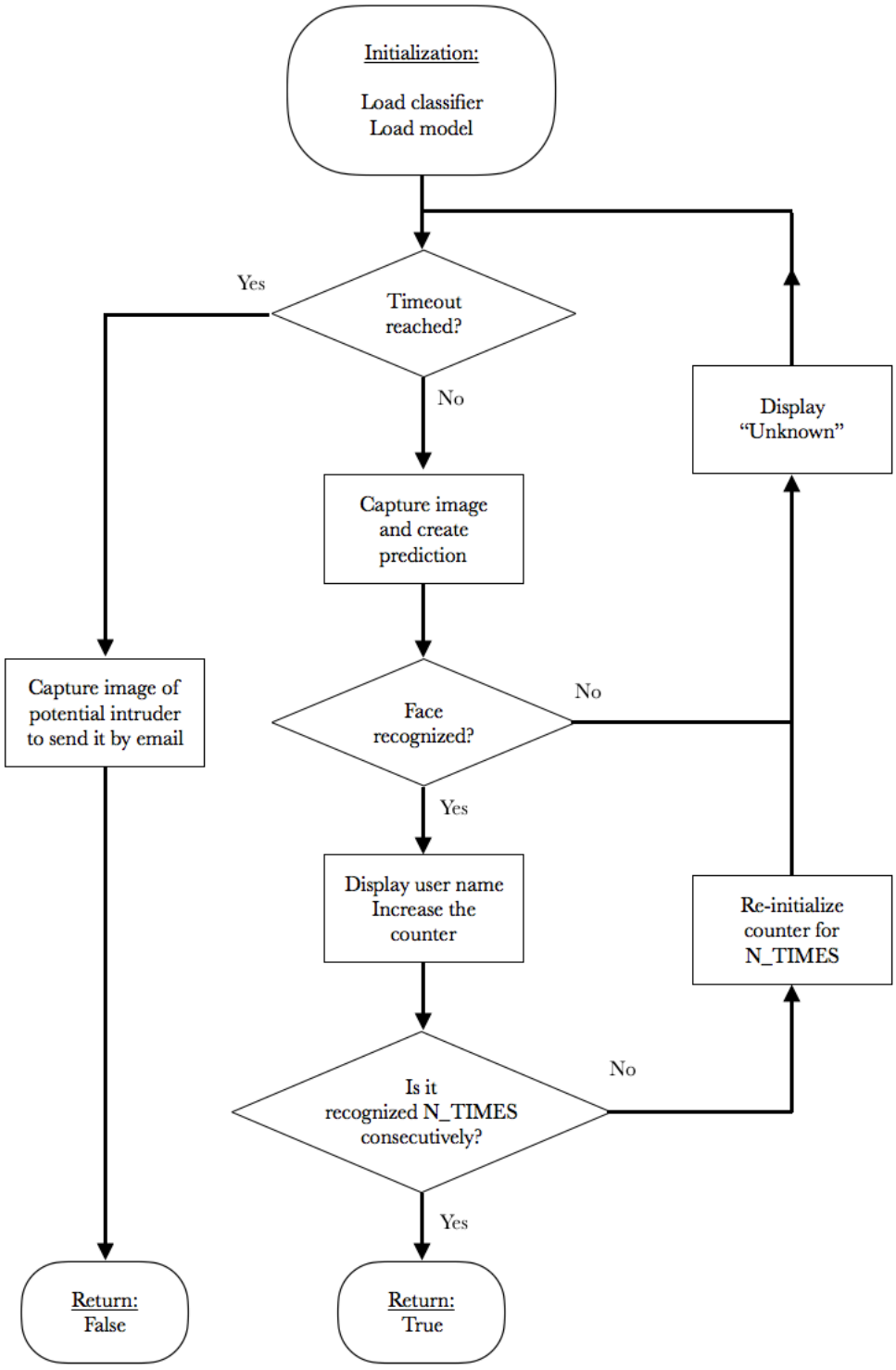


Figure 7.4: Detection stage operation diagram.

7.2 Memory, PIR, timer and email scripts

This section goes through simple but essential modules developed for the overall well-functioning of the main program. The four scripts were developed in the same period of time and they permit data management, PIR sensor interface, an effective time controlling and the online communication with the owners in case of potential intrusion.

7.2.1 Memory module

This module provides a set of functions that makes it possible to store, load and provide data in the appropriate order to the main program. There are three different types of data that need to be stored: user's name, user ID and his/her email address. All these items are modified or requested from the different phases of face recognition (see Figure 7.5). This information is stored in a .txt file called Database, which is accessed using the functions incorporated with python: open() and write(), and the attributes and methods that the returned object has, such as readlines(). There are three functions created for data management: read_users(names,ids,ems), add_user(name, id, em) and get_database().

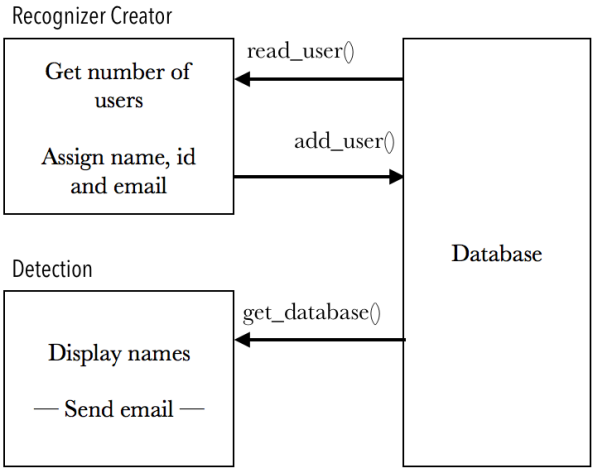


Figure 7.5: Memory access operation.

- **read_users(names, ids, ems).** Returns the number of users registered in the data base and stores the names, IDs and emails in the previously initialized names, ids and ems lists. Firstly, the Database.txt is opened and read using readlines(), which provides an array of the strings read in the text file. The multiple names, ids and email are saved

in the provided lists using a for loop that goes through every line (user) and increments the user counter every iteration.

- **add_user(name, id, em).** This function updates and adds new user information (his/her name, internally assigned ID and email) in a single line of the text file. The method write() is used for this task.

- **get_database().** Provides a python dictionary with the IDs as keys as well as the names that correspond to those IDs. This dictionary is used by the detector to display on the screen the name of the recognized user, since the prediction only provides the id and correlation. Also, in case that the recognizer is not able to link the subject face with the model trained, an email will be sent to all the email address that are in the database as a precaution measure.

7.2.2 PIR sensor interface

The PIR sensor have 3 wires to connect: vcc, ground and signal. The output of the signal is the code has to interface based on how it works. The PIR sensor itself has two slots in it, each slot is made of a special material that is sensitive to IR. The lens used here is not really doing much and so we see that the two slots can 'see' out past some distance (basically the sensitivity of the sensor). When the sensor is idle, both slots detect the same amount of IR, the ambient amount radiated from the room or walls or outdoors. When a warm body like a human or animal passes by, it first intercepts one half of the PIR sensor, which causes a positive differential change between the two halves. When the warm body leaves the sensing area, the reverse happens, whereby the sensor generates a negative differential change (see PIR datasheet for more information). Figure 7.6 shows an illustration of the signal internally generated. However, this is not the output that the Raspberry Pi will measure. Internally, the IR sensor manages mentioned signal and output a high-level output when radiation is detected and low-level output when it is not. In other words, once it takes place the differential of radiation, the PIR produces a high-level signal until it detects another (negative) differential due to the warm body disappearance. Thus, it is a pretty simple sensor to interface since it is only needed to check whether the output is active or not. This is done in python using the library RPi.GPIO provided for Raspberry Pi, and the two instructions used are initialization and the check of sensor outputs.

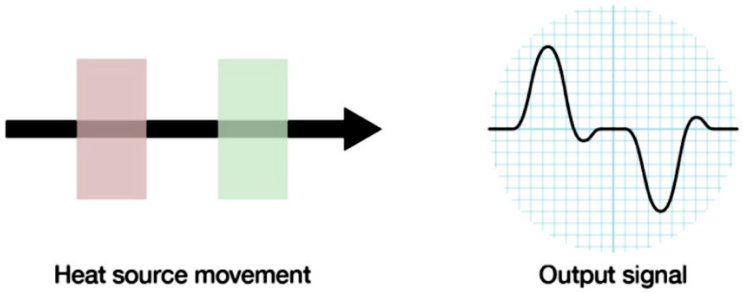


Figure 7.6: internal IR sensor output under body presence detection

7.2.3 Timer module

The timer module consists essentially on a class that permits the declaration of objects (timers) that are able to save the actual time, stablish timeouts and control the timing between two relative events in the program. It is based on the time library provided by python and permits the timer to have four basic methods.

save_time() allows the storage of current time on the attribute saved_time. update_diff_time() updates the difference of time between the saved time and the current time and saves it in the attribute limit_time. set_limit_time sets the timeout. The value specified in the fuction determines the time limit that can be used to compare two events of time. Finally, get_diff_time() is an access method that returns the current diff_time. This object is used in the detection phase to set the time limit until the detection keeps running.

7.2.4 Email module

The email script meets the set of instructions that let the main program alert all the owners that a potential intrusion could be taking place. It does two main tasks: it sends a general alert message to every member registered in the data based and secondly, it sends an image captured by the camera in the last detection iteration, so users can see who is actually trying to enter or why the detection was activated.

The different instructions used come in a python built-in package called email. All the documentation about its classes, functions, objects and values can be find in [Eldfpwe]. The class MIMEMultipart let us create an object (the message) with the body, target email address, header, etc. In order to send an email, it is necessary to have a sending email address, so it was created a gmail account for this task, although it works with any other kind of server. The server is loaded using the

smtplib libreary that is included in python as well. Once the sign-up process has been done successfully, the message previously created is attached and sent. The main program calls the send_email function n_users times, one for each owner.

7.3 Encryption and communication modules

It will be discussed the approach to get a successful integration with the SmartFusion, creating a system constituted by two different microprocessors that remains as secure as how the SmartFusion previously was by itself.

7.3.1 XTEA Encryption

Although the connection between the Raspberry Pi and the SmartFusion is wired with little chance of being jammed, it was considered appropriate to include some sort of encryption or an additional way to secure the connection between the two modules, since the overall function of the whole system relies in having a completely safe and reliable operation.

The Raspberry Pi needs to communicate with the SmartFusion to indicate whether it should open the door or not. This is basically a binary output. A potential intruder could somehow manipulate and change the output to high level much more easily than producing a sequence of high and low signals input, at the appropriate rate and with appropriate bytes. Thus, the encryption enhances exponentially the security of the whole communication. There are multiple ways to encrypt a message, but this application in particular does not require a complex encryption since the connection is wired (this did not occur with the Lora transceivers and the hub when the RSA encryption was implemented).

After doing some research, it was decided in the third milestone that XTEA (the extended version of the Tiny Encryption Algorithm) was a suitable to be implemented since it has all the features required for this part. The ciphering is made in a few lines of code (so it is not complex) but in contrast it has shared keys (in cipher and decipher). XTEA is a 64-bit block cipher with a 128-bit key and a suggested 64 rounds that includes a relatively complex key-schedule and a rearrangement of the shifts, XORs, and additions.

There are several ways to implement it in the system, for example enciphering the 64 bits and send them all to the SmartFusion or just ciphering the first 32 bits and send them to the SmartFusion which will know in advance the last 32 bits (which act as another key). This last approach is used and can be combined with

other techniques such as sending variable encrypted messages (based on even/odd numbers or prime numbers) or fixed number directly. For this application a fixed OPEN_DOOR number is sent to open the door and also a DONT_OPEN fixed number is sent to keep it unlocked.

The first step in the implementation was the testing of the C code source released into the public domain by David Wheeler and Roger Needham [WN94]. The test was simple:

```
#include <stdio.h>
#include <stdint.h>
#define ROUNDS 64

void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]);
void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]);

int main(void) {

    uint32_t mes1[2]={0x03,0xA3};
    unsigned int num_rounds = ROUNDS;
    uint32_t const key[4] = {0x34AE,0x44FB,0x19CD,0xAF19};
    printf("Message before being sent: %u %u\n",mes1[0],mes1[1]);
    encipher(num_rounds, mes1, key);
    printf("Message encrypted: %u %u\n",mes1[0],mes1[1]);
    decipher(num_rounds, mes1, key);
    printf("Message decrypted: %u %u\n\n",mes1[0],mes1[1]);

    return 0;
}


//This standard C source code, adapted from the reference code released into the
public domain by David Wheeler and Roger Needham, encrypts and decrypts using
XTEA:
/* take 64 bits of data in v[0] and v[1] and 128 bits of key[0] - key[3] */

void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], sum=0, delta=0x9E3779B9;
    for (i=0; i < num_rounds; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
        sum += delta;
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
    }
    v[0]=v0;
    v[1]=v1;
}

void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], delta=0x9E3779B9, sum=delta*num_rounds;
    for (i=0; i < num_rounds; i++) {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
        sum -= delta;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
    }
    v[0]=v0;
    v[1]=v1;
}
```

The same test was carried out in python, but with some changes. Python is a high-level programming language and its variables are not simple variables with a fixed sized like occurs in C. Python works with objects that can dynamically assign the memory needed for the value to be assigned:

```
>>> i = 1000000
>>> type(i)
<type 'int'>
>>> i = i**i
>>> type(i)
<type 'long'>
>>>
```

Therefore, the algorithm does not work as simple as in C, and from this perspective there are two approaches that can work: the conversion of the C functions into python or by using a python library that includes c types of variables.

Firstly, it was tried to import the C functions as a python library. The steps are simple: compile the library encryption.c that includes the cipher and decipher with gcc through the terminal, and import the python module "ctypes" that let us include the compiled file as a python library using the method LoadLibrary(). It only needs the path to the .so file that was generated during the compilation and once it has been imported, it is possible to use the functions included in the C library. The downside of this approach: it does not work when non-standard variables come into play. In this case, the type of variable used are imported from the <stdint.h> library, and when python tries to work with uint32_t or uint8_t it simply releases an error message.

Thus, the only way to solve this problem is to import the numpy library and use the ctype variable conversion functions that it provides. The key to get the code working is to cast every of the terms involved in the algorithm. It displays some errors regarding to overflows, but it accomplishes what wanted: the variables work as real uint32_t types so the XTEA works the same way as in C language.

7.3.2 Communication between Raspberry Pi and SmartFusion

The reason why UART communication is implemented instead of using a simple 1 or 0 output was pointed in previous section: it increases the security of the whole system not only because it avoids the use of a simple high or low signal but also provides the possibility of implementing an encryption in the communications between both devices.

The process of sending information from the Raspberry Pi to the SmartFusion basically consist in three phases, that can be seen graphically in Figure 7.7. In the first place, the information about whether the door should be unlocked or not is generated, encrypted and transformed into a string message in the Raspberry Pi side. Secondly, the string (made of the ASCII number characters) is sent through the UART connection to the SmartFusion. Lastly, the SmartFusion process the information that was sent: transforms the string into the initial uint32_t number and deciphers it. Depending on the response, the system unlocks the door or not.

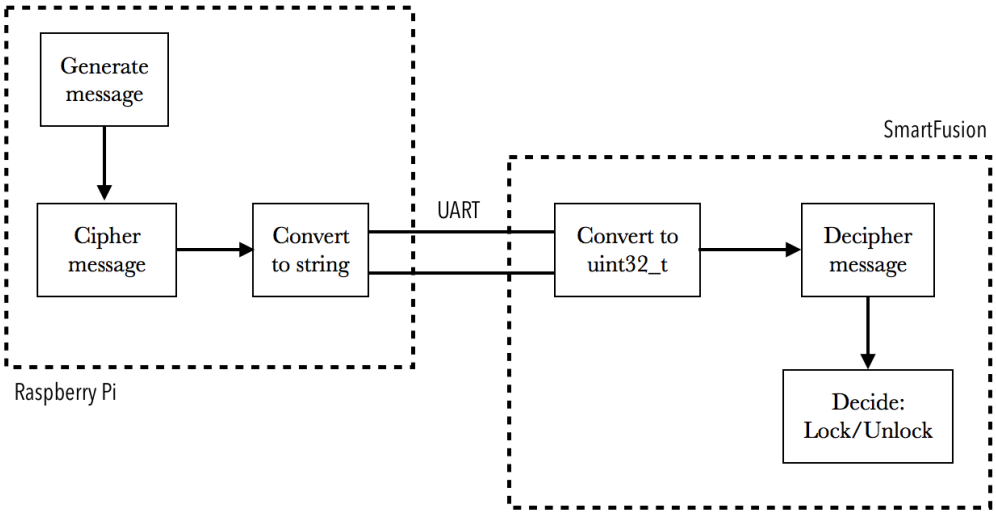


Figure 7.7: Principle of communication between the RPi and SF

The characteristics of the UART channel are standard; the information is sent at a 57600 baudrate, with no parity and one stop bit out of eight information bits. The way the uart connection is implemented for both boards is described below.

Python includes the library serial which allows an easy use of the serial port. First thing to do is the initialization, specifying the all parameters that were previous mentioned. The port to be specified is '/dev/serial0'; '/dev/ttyACM0' or '/dev/ttyUSB0' do not work since they refere to the actual USB ports in the Raspberry. Serial0 enables the Tx and Rx GPIOs (14 and 15, respectively) in the Raspberry for sending and receiving data. After enabling the port, it is possible to send data using the methods write() and receive data with readline(). The conversion from uint32_t to string is simply done with a python cast.

The SmartFusion MSS provides a set of UART drivers in addition to the SPI and IC2 seen in Part I. To initialize the UART pins the g_mss_uart1 is used intead

of g_mss_uart0, since the last one refers by default to the serial connection between the microcontroller and the PC. After initialization, the GPIOs as the UART pins, the program first reads the buffer, and if the message has been sent, it stores the sequence of characters in the memory address specified by *buff and returns a size_t variable type with a positive value. This variable is used to determine if a message was received. The asynchronous serial sometimes gets the message disorganized so it is necessary to write a function that rearrange the terms of the message (based on the fact that the last character is '\n'). This is accomplished by rearrange(), which returns 0 if the message was valid and rearranges correctly the locations of the buffer array. Once the cleared message is stored in memory, the program proceeds with its conversion to uint32_t using the string_to_uint32() function, which incorporates an iterative switch case that is combined with a pow function to sum all the digits multiplied by the correct exponent in order to get the desired number. Finally, the number converted to uint32_t is plugged into the decipher, which provides the value that was originally created. If the value corresponds to the OPEN_DOOR value, the SmartFusion will unlock the door. The main function of the library created to interface the Raspberry is FaceRecognition. It returns 0 if the face was detected, 1 if not, 2 in case there was a failure in the communication and 3 in case there was no message sent by the Raspberry Pi, and uses all the functions mentioned above.

Chapter 8

Results and conclusions

This is the last chapter regarding to technical development in this project. Here are explained the goals accomplished and steps that were necessary to get the modules developed in this part working fully integrated within the entire system.

One of the main concerns of this work is the actual reliability of the face recognition algorithm based on the minimum correlation parameter specified inside the detection process. Some tests were carried out in order to determine whether the face recognition is a reliable method to open the door (i.e. how many times the recognizer repeats without failure). The tests are wrapped up in section 8.1, in which different graphs are provided to compare how the correlation parameter and number of samples impact on the proper face recognition.

The main part is the final integration functioning. The approaches to interface the different parts of the system were discussed in the previous chapter. Once the components are all combined, the microcontrollers should assign priorities and instructions that state when the modules should be operating: how often, when and what are the connections with the other parts integrated within the system. The working flow and operation algorithm are discussed in section 8.2. Here it is possible to find how the modules were incorporated in the main.py program as well as the changes made in the main.c (incorporation of the library that interfaces the Raspberry Pi and includes the encryption algorithms).

Finally, it is discussed the goals achieved during this second part of the project, which are presented in section 8.3, as well as further work that the final product would need to include in order to meet the state-of-the-art characteristics (section 8.4).

8.1 Face recognition tests

Three different tests were carried out to adjust correctly the number of photos taken for the model and the correlation of the recognizer: one subject modifying the correlation, one subject modifying the number of pictures takes and multiple subjects with variable correlation. All the data regarding to these tests can be found in appendix C.

8.1.1 One subject with adjustable correlation

These tests were carried out with 20 samples. The results are 0 or 1 (1 means the face was recognized, 0 if the algorithm detected it as unknown). Each iteration lasts 10 seconds and evaluates if the recognition was successful or not. Figure 8.1 presents the percentage of successful tests, given the fixed correlation.

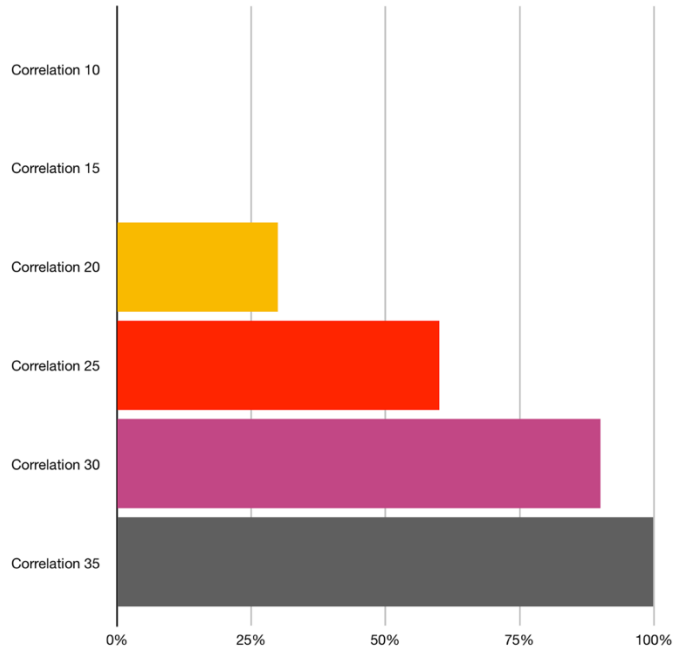


Figure 8.1: Adjustable correlation face recognition test.

8.1.2 One subject with adjustable number of samples

This test evaluates how the number of samples taken impacts of the face recognition task. The correlation is fixed to 20 (based on the previous test this is the starting point to detect faces). Figure 8.2 presents the percentage of successful tests, given a fixed number of samples.

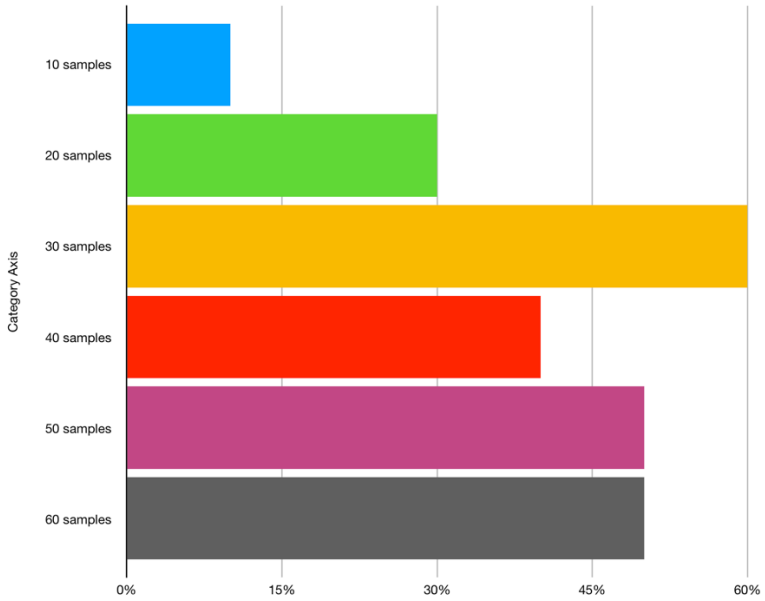


Figure 8.2: Adjustable number of samples test.

As we can see there is an important step from taking 20 samples to 30, but the rest of tests seem to produce outputs depending more on the correlation assigned than the number of samples taken. Therefore, 30 it is considered the appropriate number to create the models for face recognition in following tests.

8.1.3 Multiple subjects with variable correlation

This test focuses on multiple faces recognition. The fact that a new user is added to the equation can produce a new type of error (when the program confuses one face with another). Therefore, there is a change in notation: success (recognizes the right face), fail (recognizes the wrong face) and unknown (any face is recognized). The results are shown in Figure 8.3.

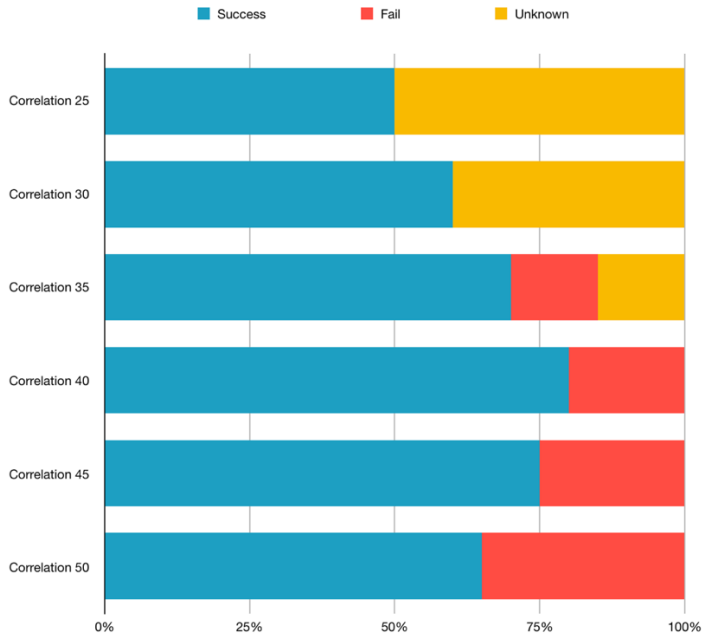


Figure 8.3: Test with multiple subjects.

8.1.4 Overall tests results

The gathered data shows different elements to analyze. In the first place: the influence of the number of face samples to obtain a reliable face recognition. When tests are carried out with a low number of pictures, there is a noticeable impact on the effectiveness of the recognizer: there are undoubtedly more “unknown” results than detections. However, as the number increases there is a point in which the recognizer no longer enhances its ability to recognize people and as result, more memory is used with no reason. 30 photos seem to be enough to achieve an optimal face recognition.

Another factor to point out is the influence of an appropriate data generation. It was noticed during the tests that the more expressions shown during the images taking, the better worked the face recognizer. For example, in the third test, subject 1 was more expressive during the data generation stage, and it possible to see that the recognizer detects much faster (i.e. with lower correlation) subject 1 than subject 2.

The last point to emphasize is the probability of the recognizer to make a mistake. Here are two statistical terms that are essential to understand the different types of potential errors:

Type I error consist in rejecting the null hypothesis when it is actually true. In our case study it is the error when a user is identified as "Unknown" instead of the stored profile. On the other hand, *Type II error* consists in not rejecting the null hypothesis when the alternate hypothesis is actually true. This means that the recognizer could identify a stranger as the real owner of the house, which is the most dangerous mistake that can occur. In the multiple faces test, type II error begins to occur when the correlation is set to 35 and higher (subject 2 starts to be recognized as subject 1) and this cannot be tolerable.

Due to these results, the number of face samples to take is determined to be 30 and minimum correlation required 30, which ensures an optimal recognition with a very low probability of committing a type II error while remaining a decent detector.

8.2 Final integration: main.py and main.c

The library incorporated to the main.c program provides two essential high-level functions: the initialization of the UART connection, which is incorporated in the init_modules() function in main.c, and FaceRecognition() which returns 0 if a face was recognized. In that case, the state machine authorizes the system to unlock the door.

The main program that the Raspberry runs has different stages in order to activate the face recognition and communicate with the SmartFusion – see figure 8.1 for an illustration of the operation algorithm –. Once the program starts running, it goes over the initialization of the devices and UART communication. In addition, it offers the possibility to add new users to the data base, only the first time that the program is run. After collecting the initial data (storage of photos), it creates the model to be used in the face recognition task. Then, the actual loop starts within the while(True).

First of all, it checks if the PIR sensor detects the radiation of any potential body. If it does, it proceeds with the face detection phase. If the face is recognized, a message is generated, ciphered and sent through the UART port to the SmartFusion. On the other hand, if the face is not detected, the door-locking message is generated, ciphered and sent as well. However, there is an additional instruction: to send a warning message by email to all the owners of the house, alerting them there is a potential intruder in the house. An image of the person in front of the door is also attached to the email to provide more information. Finally, after all the checks are done, the program finishes the iteration and goes to sleep mode for the time specified in SAMPLE_TIME.

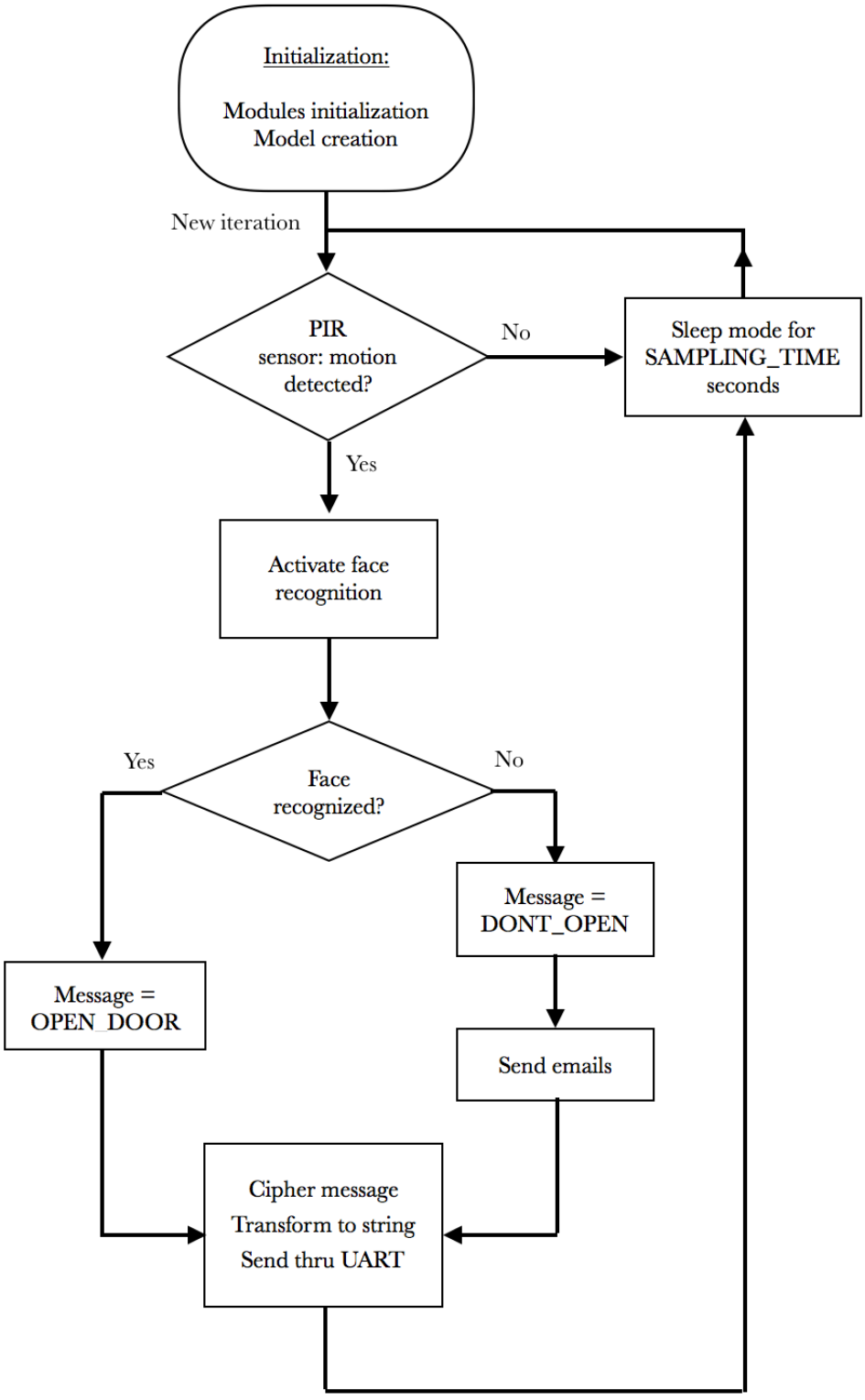


Figure 8.4: Face recognition operation

8.3 Conclusions

The integration of the whole module developed in this second part of the project was successful. Every module integrated worked perfectly in tests as well as in accordance with the other parts of the system. In addition, all the goals stated at the beginning in the first milestone were accomplished: develop an effective face recognition application, connect the module with the SmartFusion properly, and achieve a completely functional integration.

The system can work without using a device or instrument to open the door as desired. This was the last requirement of the project basic idea: to develop a reliable microprocessor-based embedded system capable of managing home entries via two-factor authentication, in which the door can be opened using a key-less sensor (NFC) as well as with no physical instrument (face recognition). The SmartFusion is connected to the internet via LoRa technology using RSA as encryption method and the Raspberry Pi via ethernet cable/WiFi. This allows remote access requests by the owners and door status checks as well as warning alerts in real time. Both modules – the Raspberry and SmartFusion – are wired using an XTEA encrypted channel.

Although the main objective has been accomplished, the product would not be in suitable conditions to be commercialized in market. It is true that the prototype is completely working, but it still is not prepared for an ordinary use. The server is built up locally so there is no access outside the local network. Also, a technician would be necessary for its installation: incorporate the NFC keys allowed, set up the server, initialize the Raspberry Pi throughout SSH tunneling, etc. There are also other elements that would be essential to incorporate in a home security system (which will be explained in next section), but unfortunately there was not enough time to develop a compact system including everything (security, reliable integration and additional implementations to make it user-friendly).

8.4 Further work

As it has been advanced in previous section, the main element remaining to enhance the development carried out is the addition of hardware and/or software that makes the system more accessible and intuitive to the ordinary user. This can be done by implemented an additional I/O screen interfaced by the Raspberry Pi or by creating a cellphone application. The main advantage of the screen is the security that it provides since it would be a wired connection. On the other hand, the

smartphone app would allow the interaction with the Raspberry Pi wirelessly and the server at the same time. This is the main reason why actually an app should be developed: to have a unique space where the server configuration can be managed (such as NFC keys authorization and add or remove keys in the server) as well as the management of the face recognition file system: add/remove users from the data base (the Raspberry Pi cannot by itself once the program has started running). Due to time constraints this was not possible to implement in the project but should be definitely done if we want to get a completely integrated system.

DOCUMENT II

MARKET STUDY

Chapter 9

Target Market

This chapter analyzes potential customers that would be interested in buying the security pack that includes all the modules discussed in the memory: NFC, LoRa technology, facial recognition, etc. The target market of the product would be Spain, a country with market characteristics that will be explain along this chapter. There are essentially two main interested parts that could obtain benefits from buying this product: security companies that are specialized in systems to secure houses, and individuals who may want to install a security system by their own. Additionally, the last section will analyze the potential impact of the prototype on the Spanish home security market as a fully developed product.

9.1 Companies

In the Spanish home security market there are two main companies that share together most of demand in the country: Securitas and Prosegur. Although there are more companies in the sector, the prototype could be interesting for any of them since both are constantly improving their security deployment and probably would like to have the most cutting-edge features as soon as possible to offer to clients. Of course, the offer would not be limited to those two companies but also the rest of the sector. However, the services that offer both companies are detailed below, in order to set the context of the market.

Currently, the services offered differ depending companies, and in most cases, they are based on deterrent measures rather than authentication features. Here are the main services provided by home security companies:

- Noisy alarm: to alert from intrusion
- Inhibitors detector: to avoid potential sabotage.

- Immediate call to the police and private security, so they are notified while the intrusion attempt takes place.
- Video camera for recording
- App control and web access to configure security equipment.
- Secondary power supply in case of black-out (electric back-up).
- NFC tags and PIN code for door opening

As can be observed, essentially all the opening features provided with the services are items already implemented in the project prototype. In fact, some of them are not included (for instance face recognition). On the other hand, the security companies provide a wide range of security sensor and actuators in case of intrusion. These elements (alarm, police call, secondary power supply, etc.) could perfectly be integrated with the system.

In conclusion, for security companies, the device developed in this project could be very interesting to incorporate to their current systems since it would be easy-to-integrate and would add new features and enhance user experience.

On the other hand, most of the newcomer domotic companies are focused on the control and the automation of house comfort variables rather than security features. This fact could also result interesting for these companies that may want to add more security features to the current project development, upgrading their offer to customers.

9.2 Individuals

Selling the product to the final customers would be also a good idea since that would reduce the final price of the product and would be more affordable than other systems from the competition.

In fact, selling the product to the final customer was the original value with which this project was born. This second target market would be people in communal living spaces such as houses or shared apartments, who are looking for greater control over the security of their own rooms/house doors by using a central system with credentials to allow access through doors as opposed to a simple key mechanism. The installation of this system is convenient for all the people that want to have a better control over the entries into the house, in addition to the

acquirement of a device that permits the access without any key, just by swapping an NFC tag or by using facial recognition.

The product is intended to reach people from all ages: from old people that might desire an easier access to their homes (no more use of physical keys) to young people that live in a communal space with other housemates and that may want to permit the access of friends remotely from the phone. Nevertheless, people within a range of age between 18 to 40 will be the target objective since the odds of acquiring new technologies are higher in the young people market.

9.3 Market niche

House demotics is booming in the Spanish market, i.e. the house owners are tending to request control and automation systems for houses: temperature regulation, light control, electric consumption management, online monitoring, cameras access, etc. Figure 9.1 shows the billing of automation and control system manufacturers over the last five years. There is no doubt: the sector of domotics continues to grow in leaps and bounds, and the security field is not completely developed; some companies offer alarms or intruders detectors but systems with NFC sensors and face recognition modules are yet to come.

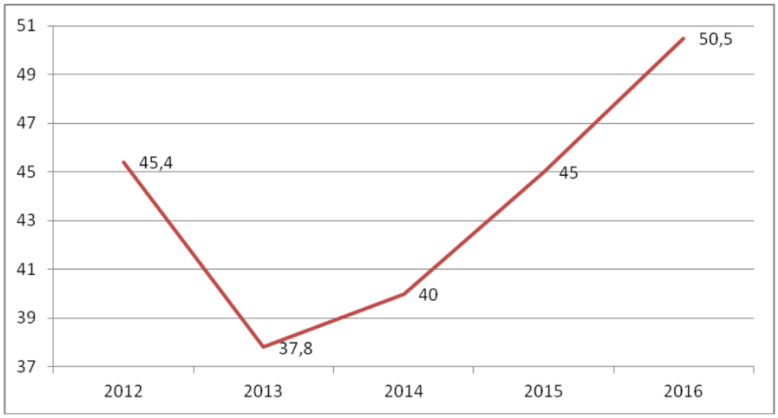


Figure 9.1: Billing of automation and control system manufacturers. Years 2012-2016 (data in millions €) [Msds].

The fact that the system enhances and focuses on the security of the house differentiates it from most of the products available at the market for that purpose, but what is more, it can be a great investment not only because it provides a more comfortable way of living but also because there is a high number of robberies and robberies attempts [Sysmi]. According to the INE (National Statistics Institute of Spain), the tendency of unarmed robberies is decreasing, but it still results in lots

of robberies in the country (for instance, 113,308 in 2016). In figure 9.2 there is a graph of the robberies occurred from 2012 to 2016.

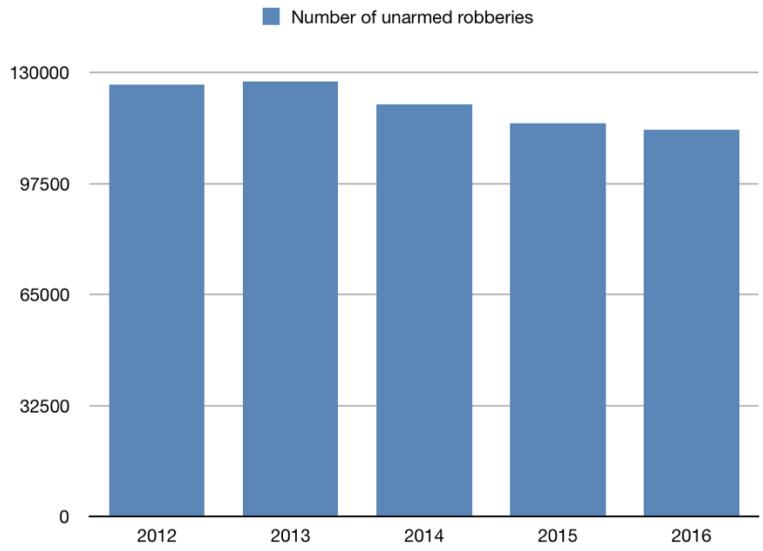


Figure 9.2: Number of unarmed robberies between 2012 and 2016 in Spain, according to the INE.

The product would not be only attractive because of the features that it comes with, but also due to the fact that it is competitive in terms of pricing. The competitors (for instance *Econergia*) offer similar security systems in a range of prices that vary from $1,500 to $23,400 [Dsb]. Something that should be said is the fact that domotic companies cover many products for temperature, lights and blinds control, rather than security, providing usually just noisy alarms and IP cameras. The prototype that was developed during the project would cost around $250 (see analysis costs in chapter 10 for more details). This means that after adding the production and installation cost it could be much more economical than products offered by the competition. Competition will be analyzed more in deep in chapter 11.

Although price is an important factor to decide between two goods, it is worth it to analyze whether the market will be willing to spend that money in domotic features (potential demand). This choice normally is limited by the available capital, which directly depends on the total family income. Because of the nature of the product (a non-essential good), it aims at people with sufficient income, i.e. the target customers that are categorized as 'lower middle class', 'upper middle class', 'upper class'. In Spain, 61,5% of the total citizens are categorized in these groups,

whose average incomes vary from $23,000 to $70,000 [G16]. This would result in a first target group of 28.6 millions of Spanish citizens. As it would be expected, the higher income would likely tend to higher number of investments.

Chapter 10

Cost Analysis

In this chapter all the costs related to the project development are presented. Firstly, a brief description of the components is provided to set the context of every device. In this section, the cost and price of every component are analyzed as well as the cost of human resources. Furthermore, there are sections that analyze the impact of producing the prototype for bulk buying as well as the number of components required depending on how big the installation is (number of doors in which the system is installed to secure the entrance).

10.1 Project components description

This section resumes all the components used in both parts of project, introducing its purpose and a brief description.

Component	Description
SmartFusion Evaluation Kit	It includes the SmartFusion Evaluation Board with SmartFusion A2F200M3F-FGG484 device, USB 2.0 A to mini-B cable and QuickStart card. Board constitutes the 'brain' of the system. Integrated in both parts of the project.
LoRa transceiver	Key device that makes it possible to connect the SmartFusion to the internet with a low power consumption through the LoRa gateway. Implemented in the first part of the project.
LoRa gateway	Also called as LG01-S or Dragino, it is an open source single channel LoRa hub. It lets bridging a LoRa wireless network to an IP network via WiFi, Ethernet, 3G or 4G cellular. Implemented in the first part of the project.

PN532 NFC sensor	NFC sensor that allows read and write tags as well as simulating NFC tags. The device offers a low-speed connection with simple setup that can be used to bootstrap more capable wireless connections. For the project, it is used to read identity documents/keycards. It was implemented in the first part of the project.
Contact switch	Magnetic contact device that is attached to the door and door frame, so the SmartFusion can know whether the door is open or not. Implemented in the first part of the project.
Hitec HS-422 - Deluxe Standard Servo	High-torque standard servo that can rotate up to 180 degrees and constitutes the actuator which opens and closes the door. Implemented in the first part of the project.
NeoPixel LED	Single LED that outputs different color lights in order to provide feedback from the current state of the door. Implemented in the first part of the project.
Raspberry Pi	Board that 1.4GHz 64-bit quad-core processor, dual-band wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and Power-over-Ethernet support. It runs Raspbian, a Linux distribution that allows the implementation of facial recognition algorithm in python. Integrated in the second part of the project.
Logitech HD Pro Webcam C920	Camera used for capturing faces from people that stand in front of the door. It is connected through the USB port and provides high definition pictures. Its features include Full HD 1080p quality video, 15 MP photo quality, H.264 encoding, Full HD glass lens technology, 20-step autofocus and built-in microphone that will not be used for the purpose of face recognition. Integrated into the SmartFusion module during the second half of the project.
PIR sensor	This sensor detects motion, and it is usually used to detect whether a human has moved in or out of the sensors range. Everything emits some low-level radiation, and the hotter something is, the more radiation is emitted, and then it is possible to know if someone is out there. Implemented during the second part of the project.

10.2 Quantity of components

The number of components needed for installation varies depending on the house and the number of doors in which the system is desired to be installed. The only exception is the gateway, which only one is necessary regardless of the number of entrance doors. Table 10.1 shows the number of elements required in a N-doors house.

Component	Quantity per house
SmartFusion Evaluation Kit	1 x N
LoRa transceiver	1 x N
LoRa gateway	1
PN532 NFC sensor	1 x N
Contact switch	1 x N
Servo	1 x N
NeoPixel LED	1 x N
Camera	1 x N
Raspberry Pi	1 x N
PIR sensor	1 x N

Table 10.1: Number of elements required per house and N doors.

10.3 Unitarian price of the modules

Table 10.2 shows the prices of each elements in addition to the online stores where they were bought.

Component	Price per unit ($)	Online store
SmartFusion Evaluation Kit	23.44	Mouser Electronics
LoRa transceiver	19.95	Adafruit
LoRa gateway	56.00	Tindie
PN532 NFC sensor	39.95	Adafruit
Contact switch	3.95	Adafruit
Servo	12.00	Adafruit
NeoPixel LED	1.00	Adafruit
Raspberry Pi	35.00	Adafruit

Camera	50.10	Amazon
PIR sensor	9.95	Adafruit
Total	247.39	

Table 10.2: Unitarian price of components.

The sum of the cost of all devices for a house with one door came to $247.39. Table 10.3 shows the total prices that would cost to install the equipment in a house with several doors. Since the average number of entrance doors varies between 1 and 2, an ordinary installation would cost $450 as much.

Doors per house	Total cost ($)
1 door	247.39
2 doors	446.68
3 doors	642.02
4 doors	837.36
5 doors	1032.70

Table 10.3: Total price installation depending on the number of doors.

Please refer to section 10.5 to see bulk buying analysis, in which prices of one door installation become cheaper with large products purchases.

10.4 Human resources costs

While in previous section it was displayed all the material cost, this section aims at estimating the human resources as well. Cost estimate[7] is displayed on table 10.4:

Profile	Hours (h)	Rate ($/h)	Total ($)
Student research work	300	37	11,100
Full professor supervision	15	175	2,625
Total			13,725

Table 10.4: Human resources cost estimation.

[7] These costs take into account both the compensation and the nominal cost of the employee [GC16].

10.5 Bulk buying analysis

This section shows the unitarian prices that is possible to obtain by making large purchases. At the end, an estimation of components bulk buying is showed. All the price information was gotten from the respective online stores.

Table 10.5 shows the unitarian price of a 50 units purchase:

Component	Unitarian price ($)
SmartFusion Evaluation Kit	19.33
LoRa transceiver	17.33
LoRa gateway	56.00
PN532 NFC sensor	35.96
Contact switch	3.55
Servo	10.80
NeoPixel LED	0.58
Camera	50.10
Raspberry Pi	35.00
PIR sensor	8.96
Total	234.69

Table 10.5: Unitarian price of components based on a 50 units purchase.

Table 10.6 shows the unitarian price of a 100 units purchase:

Component	Unitarian price ($)
SmartFusion Evaluation Kit	18.41
LoRa transceiver	15.96
LoRa gateway	56.00
PN532 NFC sensor	31.96
Contact switch	3.16
Servo	9.60
NeoPixel LED	0.51
Camera	50.10
Raspberry Pi	35.00
PIR sensor	7.96
Total	225.50

Table 10.6: Unitarian price of components based on a 100 units purchase.

Based on this data, it is possible to elaborate an estimation graph that give us an idea of the unitarian cost of the installation of a one door house. In Figure 10.1 there is a Unit price vs Units Produced graph, showing the potential cost of bulk buying implementation. As can be seen, the trend is extrapolated to obtain an approximate cost at a high production quantity. This number results in approximately in a total cost of 210 dollars.

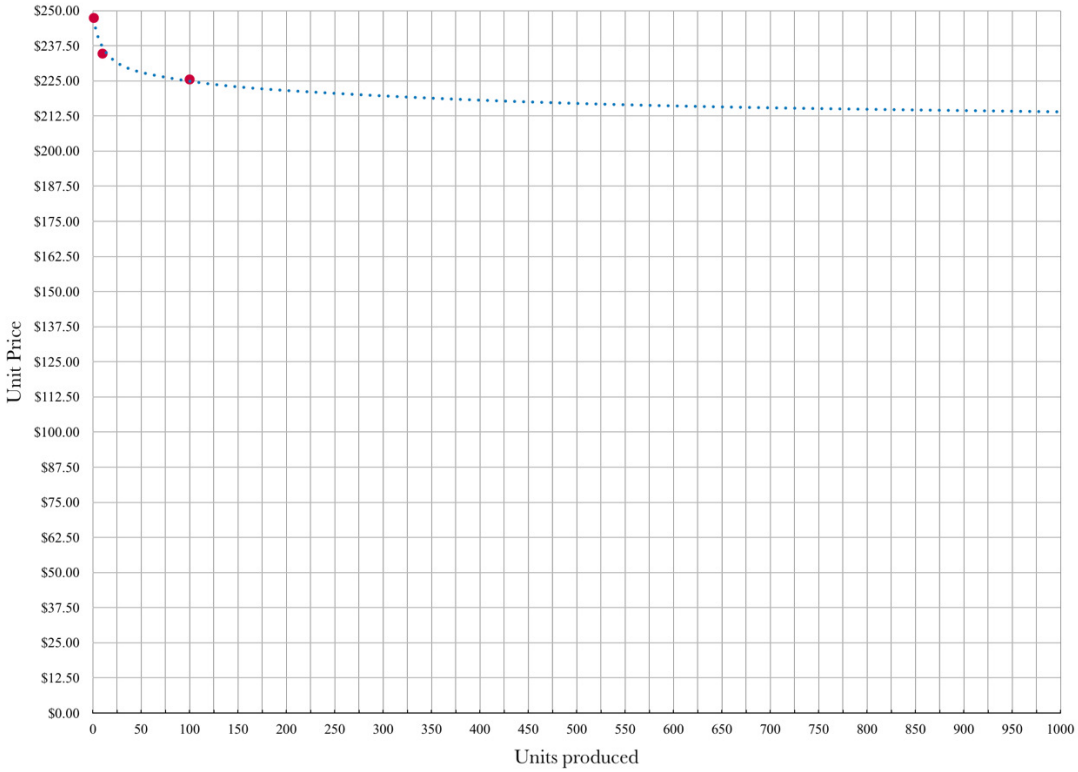


Figure 10.1: Unit prices vs. Units produced.

Chapter 11

Commercialization

Chapter 11 presents a discussion regarding to the system's final market price. In addition, some of the factors that affect to final market price such as previously studied costs and competitors' product prices are analyzed. Second section goes over different options available to place the product on market: commercialization and distribution channels. Finally, the last section provides information about possible attraction channels to engage customer to buy the access controller

11.1 Pricing and positioning

The price of the final product will be based on the previously analyzed costs, competition prices and benefits estimation.

In chapter 10 it was concluded that on a big scale it was possible to reduced costs to $210, for one door in the house. For the rest of the doors of the house the price would be around $155 since all the modules would connect to the LoRa gateway, which costs $56 and it is shared with the rest of the access controllers. This would permit the system deployment in 1, 2 or 3 entrance doors house for $210, $365 and $520. In other words, the more doors the clients install, the less money they spend in terms of product unitarian price. In particular, for the examples mentioned above it would be a material cost of $210, $182.50 and $173.33 respectively per door installed. This provides the opportunity to make a wider variety of offers to customers, in which the buyers benefit from better purchase deals in terms of unitarian cost. The estimated cost of $210 used above was obtained from the cost analysis and will be used in the sections to come as a reference of material cost. In addition, it would be necessary to consider the implicit cost of the installation as well for all the access controllers in the house. According to technical installation companies such as Lasser[8], the installation could cost around $50. Note that this cost was not included in previous analysis cost since is not a fix amount,

[8] Technical multiservice company with expertise in security equipment installations.

and it depends both on third party companies and on the size of the installation. This also happens with the transportation costs associated to the components bought, which are roughly a 10% of the total price and are discussed in the following section. However, this approximately amounts give us a relative idea of the total final cost of the installation in one door. That been said, a door installed would cost $300 total. Because competitors use products designed for a single door, this price will be used to evaluate potential pricing.

There does not exist a company that could directly compete with this kind of product. Some of them are focused on domotics and may offer security cameras but without face recognition. Others may include NFC control access systems but without remote control or face recognition. For example, companies from the security sector include services such proximation detectors, security alarms, NFC tag recognition devices, etc., but does not include a face recognition module or email notifications with intruder's picture. In other words, products in market have similar purposes but do have same features. This means that prototype designed cannot be compared with other products of companies because they do not offer the same service, but it can be compared with separated devices that offer the same features. In this case, ZKTeco keyless lock and Camera IP Wattio CAM will be analyzed in terms of available features and price and will finally be compared with the prototype to be produced on large-scale.

ZKTeco Keyless door lock was chosen because it provides similar features to the access controller: the door can be unlocked either with NFC tags, with a smartphone using Bluetooth, fingerprint and PIN code. The additional features are basically the fingerprint sensor and PIN code, which are strictly not necessary if it has a reliable NFC sensor. On the other hand, it lacks face recognition or any other system that permits the unblocking with no movements. In addition to this, it is not a reversible door lock, e.g. once it is chosen left handed or right handed cannot be installed the other way. The price of this component is $226 dollars in Amazon's marketplace.

On the other hand, Camera IP Wattio CAM could assume the role of video monitoring. It is true that it does not offer facial recognition as an implemented module, but it provides other interesting features: movement detection, night vision and listening and watching what is going on in front of the camera in real-time. This device provides similar features such as movement detection and intrusion check in case of movement detection. The main advantage is that it permits the access to the video monitoring under demand, whereas the door lock controller just sends a photo

in case the face of the person was not recognized. On the other hand, the Camera IP Wattio CAM does not provide the face recognition feature. The price of the product is 120 dollars.

Figure 11.1: ZKTeco TL400B smart door lock.

Figure 11.2: Camera IP Wattio CAM.

On the whole, although these two components offer extra features, the integration of the prototype, assembling both types of elements and features is still worth it, not only because it offers 2 main features in 1 system, but also because of its price. Table 11.1 provides a wrap-up of prices mentioned and a comparison of the total buying with the costs of the developed system.

Product	**Price ($)**	**Cost ($)**
ZKTeco Lock	226.00	
Camera IP	120.00	
Prototype		300.00
Total	346.00	300.00

Table 11.1: Comparison between competitor product prices and prototype cost.

As can be seen, the sum of both prices leads to a higher amount of money compared to the system's cost. Obviously, competitors' prices include benefits, but the difference between total cost and competitors gives already a margin of benefits: 46$ per unit. Obviously, this is an estimation and this margin varies between competitor products but is gives a general idea of price positioning.

The price setting of the product in the market place is an important decision and all the elements showed until this section should be considered: material costs, human costs, distribution costs, competitors price and benefits expectations. Based on usual sales over time distribution (Figure 11.3), the first goal would be to acquire the customer until the product gets to the maturity stage.

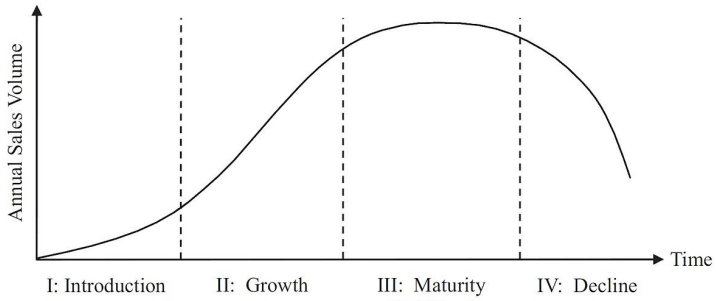


Figure 11.3: Annual sales volume curve based on stage of product.

Therefore, the product would firstly have lower prices and would rise while more customers are acquired. A reasonable price would be $350/$400 so it is affordable firstly and more people will tend to buy it. Depending on the sales volume the price will increase gradually.

Since the target market is made up of around 20 million potential custormers, achieving a sales volume of 10,000 units a year does not sound distant to reality (customer acquirement will be discussed on section 11.3) and initial investments could be returned in the next 5 years of product commercialization (this is a conservative approximation since assembling components is a variable cost).

11.2 Distribution channels

The security system could be sold to a security or domotic company that may be interested in including the particular features that this product offers into bigger infrastructures that the company has already integrated, i.e. security alarms, moves detectors, fingerprint authentication, camera monitoring, etc. The main advantage of the product would be its integrability, since it provides very specific features: extra authentication methods and remote access control.

On the other hand, the product could also be distributed through stores that are accessible to the target customer. In this case, it would be particularly beneficial to use and sell the product through online stores, not only because this approach would allow the customer to buy the product any time 24/7, but also because it would cut intermediate wholesalers and reduce costs of retailing shops. This would lead to more benefits but also would need to use an extra platform to provide external installers to start-up the device. This approach would be the most convenient for the target customers (see chapter 9) since usually this kind of products are looked up on the internet and purchased online as well.

11.3 Customer engagement marketing

In this section it is discussed the strategy to reach as many customers as possible. Since the system developed is a new concept in the domotic sector, the first step to achieve in marketplace is to get a solid base of customers. It is necessary to awaken the interest of potential customers and a good strategy to follow for its commercialization would be composed by three steps: advertising, networking and special offers.

Domotic systems are normally consulted previously on the internet before making a call to get a budget. As soon as the potential consumer starts looking up information, it becomes the appropriate moment to start displaying information about the features, price and characteristics of the product so that person has the product in mind. This can be achieved by giving priority to it in search engines such as Google, Yahoo or MSN when certain keywords are typed in the search panel (for example: security, home, NFC, face recognition, wireless, etc.). This way of advertising could also be implemented in the publicity shown in social apps such Instagram, Facebook or Twitter, which are used by young people (i.e. a great percentage of the target customers).

The use of social networks is actually really important in order to connect with the customers, receive feedback and improve services and product. Networking is essential to get promoted and recommended by customers and reach more target market.

Finally, special offers would be the perfect economic incentive to get popular within the sector of domotics. Offering free trials or making better prices with several products purchase would call the attention of interested potential customers who might end up buying more of that product or recommend it online or to friends. This would be the initial strategy to follow.

Bibliography

[MR14] V. Miori and D. Russo. "Domotic Evolution towards the IoT". In 2014 28th International Conference on Advanced Information Networking and Applications Workshops, Victoria, BC, 2014, pp. 809-814. doi: 10.1109/WAINA.2014.128

[HLMHSW16] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song and D. Wagner. Smart locks: Lessons for securing commodity internet of things devices. In Proceedings of the 11th ACM on Asia conference on computer and communications security. ACM, 2016. pp. 461-472.

[HM03] C. T. Hager and S. F. Midkiff, "An analysis of Bluetooth security vulnerabilities," 2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003., New Orleans, LA, USA, 2003, pp. 1825-1831 vol.3. doi: 10.1109/WCNC.2003.1200664

[Ogi] OpenCV library general information: https://opencv.org/about.html. Visited on: 7/6/2018.

[VM14] V. Vujović and M. Maksimović. "Raspberry Pi as a Wireless Sensor node: Performances and constraints" In 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, 2014, pp. 1013-1018. doi: 10.1109/MIPRO.2014.6859717

[Ofrd] OpenCV face recognition documentation: https://docs.opencv.org/2.4/. Visited on: 7/6/2018

[SKP15] F. Schroff, D. Kalenichenko and J. Philbin. "FaceNet: A unified embedding for face recognition and clustering" In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 815-823. doi: 10.1109/CVPR.2015.7298682

[AHP06] T. Ahonen, A. Hadid and M. Pietikainen. "Face Description with Local Binary Patterns: Application to Face Recognition". In IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.

28, no. 12, pp. 2037-2041, Dec. 2006.
doi: 10.1109/TPAMI.2006.244

[Ppld] Pillow Python library documentation: http://pillow.readthedocs.io/en/5.2.x/. Accessed: 7/6/2018

[Eofrma] Examples of OpenCV face recognition methodes and attributes: https://thecodacus.com/category/opencv/face-recognition/. Visited on: 7/6/2018

[Eldfpwe] Email library documentation for python with examples: https://docs.python.org/2/library/email-examples. Accessed: 7/6/2018

[WN94] D. J. Wheeler and R. M. Needham (1994, December). "TEA, a tiny encryption algorithm". In International Workshop on Fast Software Encryption ,pp. 363-366. Springer, Berlin, Heidelberg.

[Fbhfr] Facefirst. A brief history of face recognition: https://www.facefirst.com/blog/brief-history-of-face-recognition-software/. Visited on: 7/16/18.

[VK15] Vanlalhruaia, R. Chawngsangpuii and Yumnam Kirani Singh. "Different Approaches to Face Recognition". Vol. 4 Issue 9, September 2015. International journal of engineering research and technology.

[Rld] RadioHead library documentation: http://www.airspayce.com/mikem/arduino/RadioHead/classRH__RF95.html. Visited on: 7/9/2018

[HSS00] M. K. Hani, Tan Siang Lin and N. Shaikh-Husin. "FPGA implementation of RSA public-key cryptographic coprocessor" 2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No.00CH37119), Kuala Lumpur, 2000, pp. 6-11 vol.3. doi: 10.1109/TENCON.2000.892209

[KP11] Sushanta Kumar Sahu and Manoranjan Pradhan. "FPGA Implementation of RSA Encryption System". International Journal of Computer Applications (0975 – 8887). Volume 19– No.9, April 2011.

[Msds] Market study of domotic sector made by Cedom: http://www.cedom.es/sala-de-prensa/noticias-sector-cedom/cedom-publica-su-estudio-de-mercado-2016/download/estudio-de-mercado-2016-1-pdf. Visited on: 7/9/2018.

[Sysmi] 2016 Statistical Yearbook of the Spanish Ministry of the Interior: http://www.ine.es/prodyser/pubweb/anuario16/anu16_indigen.pdf. Visited on: 7/9/2018.

[Dsb] Domotica Integrada and Ecoenergía. Domotic solutions budgets: https://domoticaintegrada.com/presupuesto-de-domotica/ & https://www.econergia.es/kit-domotica/. Visited on: 7/16/18.

[G16] Gisbert, Francisco J. Goerlich. "Distribución de la renta, crisis económica y políticas redistributivas". Fundacion BBVA, 2016.

[GC16] José Javier Gonzalez and Kevin J. Compton. "A Simple Power Analysis Attack on the Twofish Key Schedule". pp. 22-23. University of Michigan, 2016.

[Iimcb] Infosec Institute. The most controversial biometric of all-facial recognition: https://resources.infosecinstitute.com/controversial-biometric-facial-recognition/. Visited on: 07/23/2018

[PK15] P. H. Patil and A. A. Kokil, "WiFiPi-Tracking at mass events". 2015 International Conference on Pervasive Computing *(ICPC)*, Pune, 2015, pp. 1-4. doi: 10.1109/PERVASIVE.2015.7087170

[CL13] Cheng Cai and Jianqiao Li. "Cattle Face Recognition Using Local Binary Pattern Descriptor". Department of Computer Science, College of Information Engineering, Northwest A&F University. 2013

APPENDICES

Appendix A

Here can be found all the source code. The appendix is ordered starting with the main file and libraries in C, followed by the code written in Verilog and finally the python code. At the end of the appendix it is possible to find code that cannot be classified in previous languages.

A.1 main.c

```
#include <stdio.h>
#include <inttypes.h>
#include "drivers/mss_i2c/mss_i2c.h"
#include <drivers/mss_gpio/mss_gpio.h>
#include <drivers/mss_uart/mss_uart.h>
#include "lock.h"
#include "neopixel.h"
#include "servo.h"
#include "nfc.h"
#include "contact_switch.h"
#include "lora.h"
#include "lora_client_ex.h"
#include "rsa.h"
#include "unit_tests.h"
#include "keys.h"
#include "RPI_FR.h"

#define ACCESS_GRANTED 0xA6
#define ACCESS_DENIED 0xAD

const uint32_t RSA_PUB[2] = {0, RSA_PUBLIC_KEY};
const uint32_t RSA_MOD[2] = {RSA_MODULUS_U, RSA_MODULUS_L};
const uint32_t RSA_PRI[2] = {RSA_PRIVATE_KEY_U, RSA_PRIVATE_KEY_L};
const uint32_t RSA_RES[2] = {RSA_RESIDUE_U, RSA_RESIDUE_L};
const uint32_t RSA_SERVER_MOD[2] = {RSA_SERVER_MODULUS_U,
RSA_SERVER_MODULUS_L};
const uint32_t RSA_SERVER_RES[2] = {RSA_SERVER_RESIDUE_U,
RSA_SERVER_RESIDUE_L};

//uint8_t RSA_result_ready = 0;
int LOCK_timer = 0;

// Interrupts Handler
__attribute__ ((interrupt)) void GPIO9_IRQHandler( void ){
    MSS_GPIO_clear_irq(MSS_GPIO_9);
    LORA_handle_interrupt();
}

__attribute__ ((interrupt)) void GPIO1_IRQHandler( void )
{
    MSS_GPIO_clear_irq(MSS_GPIO_1);
    nfc_set_interrupt_handled(0);
}

int main(void);
```

```
void init_modules(void);
void state_machine_door(void);
void TEST_state_machine(void);
void handle_lora_response(uint8_t* lora_buf);
void verify_nfc_list(uint8_t* nfc_response);
void unlock_door(void);
uint8_t lock_door(void);

// Main program
int main(void)
{
    MSS_GPIO_init();

    init_modules();

    TEST_state_machine();

    return(0);
}

void init_modules(void){
        MSS_GPIO_init();
        RSA_init();
        NP_init();
        LORA_init();
        CS_init();
        SERVO_init();
        nfc_init();
        RPi_init();
}

void TEST_state_machine(void){
  //uint8_t NFC_BLUE_ID[4] = {0x9B, 0x46, 0x0E, 0x89};
  //uint8_t NFC_CARD_ID[4] = {0xA8, 0x56, 0x10, 0x00};
  nfc_InListPassiveTarget();
  uint8_t lora_buf[RH_RF95_MAX_MESSAGE_LEN];
  uint8_t lora_len = sizeof(lora_buf);
  uint32_t gpio9 = (MSS_GPIO_get_inputs() & MSS_GPIO_9_MASK) >> 9;

    while(1){
        gpio9 = (MSS_GPIO_get_inputs() & MSS_GPIO_9_MASK) >> 9;
        if (CS_get_door_status() == CS_DOOR_OPEN){
            NP_set_pixel(NP_COLOUR_YELLOW);
            NP_apply();
        }else{

            if (LORA_wait_available_timeout(1000)){
                if (LORA_recv(lora_buf, &lora_len)){
                    handle_lora_response(lora_buf);
                }
            }

            if(!nfc_is_interrupt_handled()){
                nfc_read_response();
                if(!nfc_last_was_ack()){
                    uint8_t* nfc_response = nfc_get_response_buffer();
                    verify_nfc_list(nfc_response);
                    nfc_InListPassiveTarget();
                }
            }
```

```
            if(CS_get_door_status() == CS_DOOR_CLOSED && !LOCK_timer){
                SERVO_lock();
                NP_set_pixel(NP_COLOUR_BLUE);
                NP_apply();
            }

            if(FaceRecognition()==0)
                unlock_door();
        }
        if (LOCK_timer > 0){
            LOCK_timer--;
        }
    }
}

void handle_lora_response(uint8_t* lora_buf){
  uint32_t resp_msg[2];
  uint32_t decr_msg[2];
  resp_msg[0] = (lora_buf[0] << 24) | (lora_buf[1] << 16) |
    (lora_buf[2] << 8) | (lora_buf[3]);
  resp_msg[1] = (lora_buf[4] << 24) | (lora_buf[5] << 16) |
    (lora_buf[6] << 8) | (lora_buf[7]);

  RSA_run(resp_msg, RSA_PRI, RSA_MOD, RSA_RES);
  while (!RSA_is_ready());
  RSA_read_result(decr_msg);

  uint8_t access = decr_msg[1] & 0xFF;
  if (access == ACCESS_GRANTED){
    unlock_door();
  } else if (access == ACCESS_DENIED){
    lock_door();
  } else {
      NP_set_pixel_c(0xFF, 0xFF, 0xFF);
  }
}

void unlock_door(void){
  SERVO_unlock();
  NP_set_pixel(NP_COLOUR_GREEN);
  NP_apply();
  LOCK_timer = 10;
}

uint8_t lock_door(void){
  uint8_t resp = 0;
  if(CS_get_door_status() == CS_DOOR_CLOSED){
    resp = 1;
    SERVO_lock();
  }
  NP_set_pixel(NP_COLOUR_RED);
  NP_apply();
  LOCK_timer = 10;
  return resp;
}

void verify_nfc_list(uint8_t* nfc_response){
  int i;
  uint8_t nfc_id[4];
  for (i = 17; i >= 14; --i){
    nfc_id[i - 14] = nfc_response[i];
```

```
  }
  uint32_t nfc_msg[2];
  uint32_t enc_msg[2];
  nfc_msg[0] = 0;
  nfc_msg[1] = (nfc_id[0] << 24) | (nfc_id[1] << 16) |
    (nfc_id[2] << 8) | (nfc_id[3]);
  RSA_run(nfc_msg, RSA_PUB, RSA_SERVER_MOD, RSA_SERVER_RES);
  while (!RSA_is_ready());
  RSA_read_result(enc_msg);
  uint8_t enc_msg_b[8];
  for (i = 0; i < 8; ++i){
    enc_msg_b[i] = (enc_msg[i/4] >> ( (3 - (i % 4)) * 8)) & 0xFF;
  }
  LORA_send(enc_msg_b, sizeof(enc_msg_b));
  LORA_wait_packet_sent(0);
}
```

A.2 Libraries source code

A.2.1 nfc.c

```
#include "nfc.h"
#include "drivers/mss_i2c/mss_i2c.h"
#include <drivers/mss_gpio/mss_gpio.h>
#include <unistd.h>

//ack and nakframes
const uint8_t PN532_ACK_FRAME[] = {0x01, 0x00, 0x00, 0xFF, 0x00, 0xFF,
0x00};
const uint8_t PN532_NACK_FRAME[] = {0x01, 0x00, 0x00, 0xFF, 0xFF, 0x00,
0x00};
const uint8_t PN532_TURN_ON_RF_INFO[]= {0x01, 0x01};
const uint8_t PN532_POWERDOWN_INFO[] = {0x88, 0x01};
const uint8_t PN532_INLISTPASSIVETARGET_INFO[] = {0x01, 0x00};
const uint8_t PN532_SAMCONFIG_INFO[] = {0x01, 0x14, 0x01};

uint8_t last_command_sent;

uint8_t NFC_ACK_BUF[7];
const uint8_t NFC_ACK_LENGTH = 7;
uint8_t NFC_RESPONSE_BUFFER[100];
const uint8_t NFC_RESPONSE_BUFFER_LENGTH = 100;


volatile uint8_t interrupt_handled = 1;
uint8_t last_was_ack = 0;

/*
 * nfc.c
 *
 *  Created on: Nov 17, 2017
 *      Author: peces
 */


/* Function for setting up the NFC module
 * Description:
 * Receives: Nothing
 * Returns:        Nothing
```

```
 */
void nfc_init(void){
      MSS_I2C_init(&g_mss_i2c1 , PN532_I2C_ADDRESS,
MSS_I2C_PCLK_DIV_960 );
      MSS_GPIO_config( MSS_GPIO_0, MSS_GPIO_OUTPUT_MODE);
      MSS_GPIO_config( MSS_GPIO_1, MSS_GPIO_INPUT_MODE |
MSS_GPIO_IRQ_EDGE_NEGATIVE );
      MSS_GPIO_enable_irq(MSS_GPIO_1);
      int i;

      // Delays required for set up
      MSS_GPIO_set_output(MSS_GPIO_0 ,(uint8_t) 1);
      for(i=0;i<100000;i++);
      MSS_GPIO_set_output(MSS_GPIO_0 ,(uint8_t) 0);
      for(i=0;i<3000000;i++); // We need 400 milisecs (we get 418 msec =>
OK)
      MSS_GPIO_set_output(MSS_GPIO_0 ,(uint8_t)  1);
      for(i=0;i<100000;i++); //We need 10 msecs (we get 18 msec => OK)
      /* times tested with oscilloscope */
      for(i = 0; i < NFC_RESPONSE_BUFFER_LENGTH; ++i){
       NFC_RESPONSE_BUFFER[i] = 0;
      }
      nfc_SAMConfig();
      while(interrupt_handled);
      nfc_read_response();
      while(interrupt_handled);
      nfc_read_response();
      return;
}



/* NFC read function using I2C
 * Description:
 * Receives: buff -> Buffer (address) where the written data will be
stored
 *            n      -> Number of bytes to read
 *            read_ack -> if (read_ack == 1) the ack frame will be read
in before the information frame.
 * Returns:   0 if read is successful, 1 if read is not successful
 */

uint8_t nfc_read_ack(){
      MSS_I2C_read
       (
                    &g_mss_i2c1,
                    PN532_I2C_ADDRESS,
                    NFC_ACK_BUF,
                    NFC_ACK_LENGTH,
                    MSS_I2C_RELEASE_BUS
       );
      MSS_I2C_wait_complete(&g_mss_i2c1, MSS_I2C_NO_TIMEOUT);
      int i = 0;
      int isAck = 1;
      int isNack = 1;
      for(i = 0; i < NFC_ACK_LENGTH; ++i){
       if (NFC_ACK_BUF[i] != PN532_ACK_FRAME[i])
             isAck = 0;
       if (NFC_ACK_BUF[i] != PN532_NACK_FRAME[i])
             isNack = 0;
      }
```

```
	if(!isAck || isNack)
	 return 0;
	return 1;
}

uint8_t nfc_read_response(){
	nfc_set_interrupt_handled(1);
	if(last_was_ack){
	 int i = 0;
	 for(i = 0; i < NFC_RESPONSE_BUFFER_LENGTH; ++i){
		NFC_RESPONSE_BUFFER[i] = 0;
	 }
	 int count = 0;
	 while(NFC_RESPONSE_BUFFER[0] != 0x01 && count < 10){
		MSS_I2C_read
			(
				&g_mss_i2c1,
				PN532_I2C_ADDRESS,
				NFC_RESPONSE_BUFFER,
				nfc_get_response_size(),
				MSS_I2C_RELEASE_BUS
			);
		MSS_I2C_wait_complete(&g_mss_i2c1, MSS_I2C_NO_TIMEOUT);
		++count;
	 }
	 last_was_ack = 0;
	 return 0;
	} else {
	 if(nfc_read_ack()){
		last_was_ack = 1;
		return 1;
	 }else{
		return 0;
	 }
	}
}

uint8_t *nfc_get_response_buffer(){
	return NFC_RESPONSE_BUFFER;
}


/*
 * Description:
 * Receives:
 * Returns:   Nothing
 */
void nfc_send_command(const uint8_t command, const uint8_t info_buf[],
const uint8_t len){
	uint16_t length = 0;
	uint8_t buffer[20];
	buffer[length++] = (uint8_t) PN532_PREAMBLE; // Preamble
	buffer[length++] = 0x00; // Start bits
	buffer[length++] = 0xFF; // Start bits
	buffer[length++] = (len + 2); // Length of (n_data_bytes + TFI byte
+ command)
	buffer[length++] = (uint8_t)(0x100 - (len + 2)); // LCS
	buffer[length++] = (uint8_t) PN532_HOST_TO_NFC; // Write
	uint8_t dcs = (uint8_t)PN532_HOST_TO_NFC + command;
	buffer[length++] = command;
	int i = 0;
```

```
      for(i = 0; i < len; ++i){
       dcs += info_buf[i];
       buffer[length++] = info_buf[i];
      }
      buffer[length++] = (uint8_t)(0x100 - dcs); // DCS
      buffer[length++] = PN532_POSTAMBLE; // Postamble

      MSS_I2C_write
       (
                   &g_mss_i2c1,
                   PN532_I2C_ADDRESS,
                   buffer,
                   length,
                   MSS_I2C_RELEASE_BUS
       );
      MSS_I2C_wait_complete(&g_mss_i2c1, MSS_I2C_NO_TIMEOUT);
      return;
}


/*
 * Description: Function to get all the parameters of the status of the
NFC
 * Receives: Array LENGTH 9(+8) where the info will be stored
 * Returns:   field (if external RF is present=1, 0 otherwise)
 */
uint8_t nfc_GetGeneralStatus(){
      uint8_t command[] = {};
      nfc_send_command(PN532_COMMAND_GETSTATUS, command, 0);
      last_command_sent = PN532_COMMAND_GETSTATUS;
      return 1;
}

uint8_t nfc_InListPassiveTarget(){

      //send D4 4A 01 00
      //receive D5 4B <num targets> <tg (1) sens_res (2) sel_res (1)
nfcidlength (1) nfcid (4?) atslength (1) ats (y)>

      nfc_send_command(PN532_COMMAND_INLISTPASSIVETARGET,
                           PN532_INLISTPASSIVETARGET_INFO,
                           2);
      last_command_sent = PN532_COMMAND_INLISTPASSIVETARGET;
      return 1; //response_buf[6];
}

uint8_t nfc_SAMConfig(){
  uint8_t command[] = {0x01, 0x14, 0x01};
  nfc_send_command(PN532_COMMAND_SAMCONFIG, PN532_SAMCONFIG_INFO, 3);
  last_command_sent = PN532_COMMAND_SAMCONFIG;
  return 1;// response_buf[6];
}

uint8_t nfc_GetFirmwareVersion(){
      uint8_t command[] = {};
      nfc_send_command(PN532_COMMAND_GETFIRMWARE, command, 0);
      last_command_sent = PN532_COMMAND_GETFIRMWARE;
      return 1;
}

uint8_t nfc_is_interrupt_handled(void){
```

```
      return interrupt_handled;
}

void nfc_set_interrupt_handled(uint8_t val){
      interrupt_handled = val;
}

uint8_t nfc_get_response_size(){
      switch(last_command_sent){
      case PN532_COMMAND_GETFIRMWARE: return 14;
      case PN532_COMMAND_POWERDOWN: return 5;
      case PN532_COMMAND_GETSTATUS: return 20;
      case PN532_COMMAND_INAUTOPOLL: return 5;
      case PN532_COMMAND_RF_CONFIG: return 5;
      case PN532_COMMAND_INLISTPASSIVETARGET: return 20;
      case PN532_COMMAND_SAMCONFIG: return 10;
      default: return NULL;
      }
}

uint8_t nfc_last_was_ack(){
      return last_was_ack;
}
```

A.2.1 lora.c

```
#include "lora.h"

const uint8_t frame_size = 16;
const uint8_t burst_frame_size = 8;
const uint8_t modem_default[] = {0x72, 0x74, 0x00};

volatile uint8_t buf_len;
volatile uint8_t rx_buf_valid;
uint8_t buf[RH_RF95_MAX_PAYLOAD_LEN];

// BEGIN RHGenericDriver code
uint8_t mode;

uint8_t this_address = DEVICE_ID;
uint8_t promiscuous = 0;

volatile uint8_t rx_header_to;
volatile uint8_t rx_header_from;
volatile uint8_t rx_header_id;
volatile uint8_t rx_header_flags;

uint8_t tx_header_to = GATEWAY_ID;
uint8_t tx_header_from = DEVICE_ID;
uint8_t tx_header_id = 0x01;
uint8_t tx_header_flags = 0x00;

volatile int16_t last_rssi;

volatile uint16_t rx_bad;
volatile uint16_t rx_good;
volatile uint16_t tx_good;

volatile uint8_t cad;
unsigned int cad_timeout;
```

```
void LORA_handle_interrupt(void)
{
    // Read the interrupt register
    uint8_t irq_flags = LORA_read(RH_RF95_REG_12_IRQ_FLAGS);
    //printf("LORA MODE: %u\r\n", mode);
    if (mode == LORA_MODE_RX && irq_flags & (RH_RF95_RX_TIMEOUT |
RH_RF95_PAYLOAD_CRC_ERROR))
    {
      rx_bad++;
    }
    else if (mode == LORA_MODE_RX && irq_flags & RH_RF95_RX_DONE)
    {
      // Have received a packet
       uint8_t len = LORA_read(RH_RF95_REG_13_RX_NB_BYTES);

       // Reset the fifo read ptr to the beginning of the packet
       LORA_write(RH_RF95_REG_0D_FIFO_ADDR_PTR,
LORA_read(RH_RF95_REG_10_FIFO_RX_CURRENT_ADDR));
       LORA_burst_read(RH_RF95_REG_00_FIFO, buf, len);
       buf_len = len;
       LORA_write(RH_RF95_REG_12_IRQ_FLAGS, 0xff); // Clear all IRQ flags

       // Remember the RSSI of this packet
       // this is according to the doc, but is it really correct?
       // weakest receiveable signals are reported RSSI at about -66
       last_rssi = LORA_read(RH_RF95_REG_1A_PKT_RSSI_VALUE) - 137;

       // We have received a message.
       LORA_validate_rx_buf();
       if (rx_buf_valid)
             LORA_set_mode_idle(); // Got one
    }
    else if (mode == LORA_MODE_TX && irq_flags & RH_RF95_TX_DONE)
    {
       tx_good++;
       LORA_set_mode_idle();
    }
    else if (mode == LORA_MODE_CAD && irq_flags & RH_RF95_CAD_DONE)
    {
        cad = irq_flags & RH_RF95_CAD_DETECTED;
        LORA_set_mode_idle();
    }

    LORA_write(RH_RF95_REG_12_IRQ_FLAGS, 0xff); // Clear all IRQ flags
}

void LORA_wait_available(void){
     while (LORA_available() == FALSE);
}

uint8_t LORA_wait_available_timeout(uint16_t timeout){
     unsigned long starttime = timeout * 1000;
     unsigned long counter = 0;
     while (counter++ < starttime){
      if (LORA_available() != FALSE){
            return TRUE;
      }
     }
     return FALSE;
}
```

```
uint8_t LORA_wait_packet_sent(uint16_t timeout){
      if (!timeout){
       while (mode == LORA_MODE_TX);
       return TRUE;
      }
      //unsigned long starttime = time(0);
      //while ((time(0) - starttime < timeout)){
      unsigned long starttime = timeout * 1000;
      unsigned long counter = 0;
      while (counter++ < starttime){
       if (mode != LORA_MODE_TX){
             return TRUE;
       }
      }
      return FALSE;
}

void LORA_set_promiscuous(uint8_t prom){
      promiscuous = prom;
}

void LORA_set_this_address(uint8_t addr){
      this_address = addr;
}

void LORA_set_header_to(uint8_t to){
      tx_header_to = to;
}

void LORA_set_header_from(uint8_t from){
      tx_header_from = from;
}

void LORA_set_header_id(uint8_t id){
      tx_header_id = id;
}

void LORA_set_header_flags(uint8_t set, uint8_t clear){
      tx_header_flags &= ~clear;
      tx_header_flags |= set;
}

// END RHGenericDriver code

uint8_t LORA_init(void){
     MSS_GPIO_config( MSS_GPIO_8, MSS_GPIO_OUTPUT_MODE);
     MSS_GPIO_config( MSS_GPIO_9, MSS_GPIO_INPUT_MODE |
MSS_GPIO_IRQ_EDGE_POSITIVE );
     MSS_GPIO_enable_irq(MSS_GPIO_9);
     MSS_GPIO_set_output(MSS_GPIO_8, 1);
     MSS_SPI_init(&g_mss_spi1);
     MSS_SPI_configure_master_mode
     (
      &g_mss_spi1,
      MSS_SPI_SLAVE_0,
      MSS_SPI_MODE0,
      MSS_SPI_PCLK_DIV_256,
      frame_size
     );
```

```
      NVIC_SetPriority(41, 1);
      //MSS_GPIO_drive_inout(MSS_GPIO_10, MSS_GPIO_HIGH_Z);
      int i;
      LORA_write(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_SLEEP |
RH_RF95_LONG_RANGE_MODE);
//        delay(10); // Wait for sleep mode to take over from say, CAD
      for (i=0;i<100000;i++);

      LORA_read(RH_RF95_REG_06_FRF_MSB);
      uint8_t read_result = LORA_read(RH_RF95_REG_01_OP_MODE);
      uint8_t comp = RH_RF95_MODE_SLEEP | RH_RF95_LONG_RANGE_MODE;
          // Check we are in sleep mode, with LORA set
      if (read_result != comp)
      {
       //     Serial.println(spiRead(RH_RF95_REG_01_OP_MODE), HEX);
       return 1; // No device present?
      }

      // Sets up FIFO so transmit data starts at 0, receive starts at 128
      LORA_write(RH_RF95_REG_0E_FIFO_TX_BASE_ADDR, 0);
      LORA_write(RH_RF95_REG_0F_FIFO_RX_BASE_ADDR, 128);

      LORA_set_mode_idle();

      LORA_set_modem_config(modem_default); // Radio default

      LORA_set_preamble_length(8);

      LORA_set_frequency(915.0);

      // Lowish power
      LORA_set_tx_power(13, 0);

  //LORA_set_header_from(DEVICE_ID);
  //LORA_set_this_address(DEVICE_ID);

      return 0;
}

void LORA_set_mode_idle(void){
      if (mode != LORA_MODE_IDLE){
       LORA_write(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_STDBY);
       mode = LORA_MODE_IDLE;
      }
}

void LORA_sleep(void){
      if (mode != LORA_MODE_SLEEP){
       LORA_write(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_SLEEP);
       mode = LORA_MODE_SLEEP;
      }
}

void LORA_set_mode_rx(void){
      if (mode != LORA_MODE_RX){
       LORA_write(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_RXCONTINUOUS);
       LORA_write(RH_RF95_REG_40_DIO_MAPPING1, 0x00); // Interrupt on
RxDone
```

```
       mode = LORA_MODE_RX;
      }
}

void LORA_set_mode_tx(void){
    if (mode != LORA_MODE_TX)
    {
      LORA_write(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_TX);
      LORA_write(RH_RF95_REG_40_DIO_MAPPING1, 0x40); // Interrupt on
TxDone
      mode = LORA_MODE_TX;
    }
}

void LORA_set_modem_config(uint8_t *config){
     LORA_write(RH_RF95_REG_1D_MODEM_CONFIG1, config[0]);
     LORA_write(RH_RF95_REG_1E_MODEM_CONFIG2, config[1]);
     LORA_write(RH_RF95_REG_26_MODEM_CONFIG3, config[2]);
}

void LORA_set_preamble_length(uint16_t bytes){
     LORA_write(RH_RF95_REG_20_PREAMBLE_MSB, bytes >> 8);
     LORA_write(RH_RF95_REG_21_PREAMBLE_LSB, bytes & 0xff);
}

void LORA_set_frequency(double f){
     uint32_t frf = (f * 1000000.0) / RH_RF95_FSTEP;
     LORA_write(RH_RF95_REG_06_FRF_MSB, (frf >> 16) & 0xff);
     LORA_write(RH_RF95_REG_07_FRF_MID, (frf >> 8) & 0xff);
     LORA_write(RH_RF95_REG_08_FRF_LSB, frf & 0xff);
}

void LORA_set_tx_power(int8_t power, uint8_t useRFO)
{
    // Sigh, different behaviours depending on whther the module use
PA_BOOST or the RFO pin
    // for the transmitter output
    if (useRFO){
      if (power > 14)
            power = 14;
      if (power < -1)
            power = -1;
      LORA_write(RH_RF95_REG_09_PA_CONFIG, RH_RF95_MAX_POWER | (power +
1));
    } else {
      if (power > 23)
            power = 23;
      if (power < 5)
            power = 5;

      // For RH_RF95_PA_DAC_ENABLE, manual says '+20dBm on PA_BOOST when
OutputPower=0xf'
      // RH_RF95_PA_DAC_ENABLE actually adds about 3dBm to all power
levels. We will use it
      // for 21, 22 and 23dBm
      if (power > 20) {
            LORA_write(RH_RF95_REG_4D_PA_DAC, RH_RF95_PA_DAC_ENABLE);
            power -= 3;
      } else {
            LORA_write(RH_RF95_REG_4D_PA_DAC, RH_RF95_PA_DAC_DISABLE);
      }
```

```
        // RFM95/96/97/98 does not have RFO pins connected to anything.
Only PA_BOOST
        // pin is connected, so must use PA_BOOST
        // Pout = 2 + OutputPower.
        // The documentation is pretty confusing on this topic: PaSelect
says the max power is 20dBm,
        // but OutputPower claims it would be 17dBm.
        // My measurements show 20dBm is correct
        LORA_write(RH_RF95_REG_09_PA_CONFIG, RH_RF95_PA_SELECT | (power-
5));
      }
}

void LORA_validate_rx_buf(void){
      if (buf_len < 4){
        return;
      }
      rx_header_to = buf[0];
      rx_header_from = buf[1];
      rx_header_id = buf[2];
      rx_header_flags = buf[3];
      if (promiscuous ||
              rx_header_to == this_address ||
              rx_header_to == RH_BROADCAST_ADDRESS){
        rx_good++;
        rx_buf_valid = 1;
      }
}

uint8_t LORA_available(void){
      if (mode == LORA_MODE_TX){
        return FALSE;
      }
      LORA_set_mode_rx();
      return rx_buf_valid;
}

void LORA_clear_rx_buf(void){
      rx_buf_valid = FALSE;
      buf_len = 0;
}

uint8_t LORA_send(const uint8_t* data, uint8_t len){
      if (len > RH_RF95_MAX_MESSAGE_LEN){
        return FALSE;
      }

      LORA_wait_packet_sent(0);
      LORA_set_mode_idle();
  LORA_set_header_to(GATEWAY_ID);

      // Position at the beginning of the FIFO
      LORA_write(RH_RF95_REG_0D_FIFO_ADDR_PTR, 0);
      // The headers
      LORA_write(RH_RF95_REG_00_FIFO, tx_header_to);
      LORA_write(RH_RF95_REG_00_FIFO, tx_header_from);
      LORA_write(RH_RF95_REG_00_FIFO, tx_header_id);
      LORA_write(RH_RF95_REG_00_FIFO, tx_header_flags);
      // The message data
      LORA_burst_write(RH_RF95_REG_00_FIFO, data, len);
```

```
      LORA_write(RH_RF95_REG_22_PAYLOAD_LENGTH, len +
RH_RF95_HEADER_LEN);

      LORA_set_mode_tx(); // Start the transmitter
      // when Tx is done, interruptHandler will fire and radio mode will
return to STANDBY
}

uint8_t LORA_recv(uint8_t* in_buf, uint8_t* len){
    if (!LORA_available()){
      return 0;
    }
      if (in_buf && len) {
       // Skip the 4 headers that are at the beginning of the rxBuf
       if (*len > buf_len - RH_RF95_HEADER_LEN)
             *len = buf_len - RH_RF95_HEADER_LEN;
       memcpy(in_buf, buf + RH_RF95_HEADER_LEN, *len);
    }
    LORA_clear_rx_buf(); // This message accepted and cleared
    return TRUE;
}

uint8_t LORA_read(uint8_t addr){
      uint8_t response;
      //MSS_SPI_set_slave_select(&g_mss_spi1, MSS_SPI_SLAVE_0);
      MSS_GPIO_set_output(MSS_GPIO_8, 0);
      response = MSS_SPI_transfer_frame(&g_mss_spi1, addr << 8);
      MSS_GPIO_set_output(MSS_GPIO_8, 1);
      //MSS_SPI_clear_slave_select(&g_mss_spi1, MSS_SPI_SLAVE_0);
      return response;
}

void LORA_write(uint8_t addr, uint8_t data){
      uint16_t cmd = (1 << 15) | (addr << 8) | data;
      MSS_SPI_set_slave_select(&g_mss_spi1, MSS_SPI_SLAVE_0);
      MSS_GPIO_set_output(MSS_GPIO_8, 0);
      MSS_SPI_transfer_frame( &g_mss_spi1, cmd);
      MSS_GPIO_set_output(MSS_GPIO_8, 1);
      MSS_SPI_clear_slave_select(&g_mss_spi1, MSS_SPI_SLAVE_0);
}

// TODO: make sure these can properly do burst
uint8_t LORA_burst_read(uint8_t addr, uint8_t* res, uint8_t len){
      MSS_SPI_configure_master_mode
       (
             &g_mss_spi1,
             MSS_SPI_SLAVE_0,
             MSS_SPI_MODE0,
             MSS_SPI_PCLK_DIV_256,
             burst_frame_size
       );
  uint8_t status = 0;
  MSS_SPI_set_slave_select(&g_mss_spi1, MSS_SPI_SLAVE_0);
  MSS_GPIO_set_output(MSS_GPIO_8, 0);
  status = MSS_SPI_transfer_frame(&g_mss_spi1, addr);
  int i;
  for (i = 0; i < len; ++i){
    res[i] = MSS_SPI_transfer_frame(&g_mss_spi1, 0);
  }
  MSS_GPIO_set_output(MSS_GPIO_8, 1);
  MSS_SPI_clear_slave_select(&g_mss_spi1, MSS_SPI_SLAVE_0);
```

```
  MSS_SPI_configure_master_mode
      (
       &g_mss_spi1,
       MSS_SPI_SLAVE_0,
       MSS_SPI_MODE0,
       MSS_SPI_PCLK_DIV_256,
       frame_size
      );
  return status;
}

uint8_t LORA_burst_write(uint8_t addr, uint8_t* src, uint8_t len){
     MSS_SPI_configure_master_mode
      (
       &g_mss_spi1,
       MSS_SPI_SLAVE_0,
       MSS_SPI_MODE0,
       MSS_SPI_PCLK_DIV_256,
       burst_frame_size
      );
  uint8_t status = 0;
  MSS_SPI_set_slave_select(&g_mss_spi1, MSS_SPI_SLAVE_0);
  MSS_GPIO_set_output(MSS_GPIO_8, 0);
  status = MSS_SPI_transfer_frame(&g_mss_spi1, (1 << 7) | addr);
  int i;
  for (i = 0; i < len; ++i){
    MSS_SPI_transfer_frame(&g_mss_spi1, src[i]);
  }
  MSS_GPIO_set_output(MSS_GPIO_8, 1);
  MSS_SPI_clear_slave_select(&g_mss_spi1, MSS_SPI_SLAVE_0);
  MSS_SPI_configure_master_mode
      (
       &g_mss_spi1,
       MSS_SPI_SLAVE_0,
       MSS_SPI_MODE0,
       MSS_SPI_PCLK_DIV_256,
       frame_size
      );
  return status;
}
```

A.2.3 servo.c

```
#include "servo.h"

void SERVO_init(void){
     // intentionally blank
}

void SERVO_lock(void){
     (*SERVO) = (uint32_t) SERVO_LOCKED;
}

void SERVO_unlock(void){
     (*SERVO) = (uint32_t) SERVO_UNLOCKED;
}

uint32_t SERVO_read_state(void){
     return (uint32_t) *SERVO;
```

```
    }
```

A.2.4 neopixel.c

```
#include "neopixel.h"

volatile uint32_t *NP_ADDRESS = (uint32_t*) 0x40050000;

void NP_init(void){
      // intentionally empty
}

void NP_set_pixel(uint32_t colour){
      *(NP_ADDRESS) = NP_CONTROL_SET | colour;
}

void NP_set_pixel_c(uint8_t red, uint8_t green, uint8_t blue){
      uint32_t set_colour = 0x00000000;
      set_colour |= ((uint32_t) green) << 16;
      set_colour |= ((uint32_t) red) << 8;
      set_colour |= (uint32_t) blue;
      set_colour |= NP_CONTROL_SET;
      *(NP_ADDRESS) = set_colour;
}

void NP_apply(void){
      *(NP_ADDRESS) = NP_CONTROL_APPLY;
}

void NP_clear(void){
      *(NP_ADDRESS) = NP_CONTROL_CLEAR;
}

uint32_t NP_get_pixel(void){
      return (uint32_t) *(NP_ADDRESS);

    }
```

A.2.5 contact_switch.c

```
#include "contact_switch.h"

void CS_init(void){
      MSS_GPIO_config( MSS_GPIO_4, MSS_GPIO_INPUT_MODE);
}

uint8_t CS_get_door_status(void){
      uint32_t inputs = MSS_GPIO_get_inputs();
      return (uint8_t) (inputs & MSS_GPIO_4_MASK) >> 4;

    }
```

A.2.6 rsa.c

```
#include "rsa.h"

void RSA_init(void){
      *RSA_BEGIN_ENCRYPT = 0x0;
```

```
}

void RSA_run(uint32_t* message, uint32_t *key, uint32_t *modulus,
uint32_t *residue){
      *RSA_BEGIN_ENCRYPT = 0x0;

      *RSA_BIT_SWITCH = LOAD_UPPER;
      *RSA_MESSAGE = message[0];
      *RSA_EXPONENT = key[0];
      *RSA_MODULUS = modulus[0];
      *RSA_RESIDUE = residue[0];
      *RSA_BIT_SWITCH = LOAD_LOWER;
      *RSA_MESSAGE = message[1];
      *RSA_EXPONENT = key[1];
      *RSA_MODULUS = modulus[1];
      *RSA_RESIDUE = residue[1];

      *RSA_BEGIN_ENCRYPT = 0x1;

}

uint8_t RSA_read_result(uint32_t result[2]){
      uint8_t valid = *RSA_RESULT_VALID;
      if (!valid){
       return 1;
      }
      *RSA_BIT_SWITCH = LOAD_LOWER;
      result[1] = *RSA_RESULT;
      *RSA_BIT_SWITCH = LOAD_UPPER;
      result[0] = *RSA_RESULT;
      *RSA_BEGIN_ENCRYPT = 0x0;
      return 0;
}

uint8_t RSA_is_ready(void){
      return *RSA_RESULT_VALID;

    }
```

A.2.7 RPi_FR.c

```
//
//  RPi_FR.c
//  RPi_FR
//
//  Created by Cristian on 4/22/18.
//  Copyright © 2018 Cristian. All rights reserved.
//

#include "RPi_FR.h"
#include <inttypes.h>
#include "drivers/mss_uart/mss_uart.h"
#include <drivers/mss_gpio/mss_gpio.h>
#define ROUNDS 64
#define REC_OPEN_DOOR 3
#define REC_DONT_OPEN 1
#define KEY_OPEN 1250640089
#define KEY_LOCK 3530294326
#define BUFF_SIZE 11
```

```
void RPi_init(void){
    MSS_UART_init
        (
         &g_mss_uart1,
         MSS_UART_57600_BAUD,
         MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY |
MSS_UART_ONE_STOP_BIT
         );

}

int FaceRecognition(void){ // 0->opens, 1-> locks, 2->failure in
communication, 3->message not received
    uint32_t mes1[2],mes2[2];
    size_t rx_size;
    uint32_t const key[4] = {13486,17659,6605,44825};
    unsigned int num_rounds = ROUNDS;
    int res=3;
    int isValid;
    uint8_t rx_buff[BUFF_SIZE];

    rx_size = MSS_UART_get_rx( &g_mss_uart1, rx_buff, sizeof(rx_buff));
    if (rx_size > 0){
        isValid = rearrange(rx_buff);
        if(isValid == 0){
            mes1[0] = string_to_uint32(rx_buff, BUFF_SIZE);
            mes2[0]=mes1[0];
            mes1[1]= (uint32_t) KEY_OPEN;
            mes2[1]= (uint32_t) KEY_LOCK;
            decipher(num_rounds, mes1, key);
            decipher(num_rounds, mes2, key);

            if(mes1[0] == 3){
                res = 0;
            }else if(mes2[0] == 1){
                res = 1;
            }else{
                res = 2;
            }
    }
        return res;
}


void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const
key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], sum=0, delta=0x9E3779B9;
    for (i=0; i < num_rounds; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
        sum += delta;
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) &
3]);
    }
    v[0]=v0; v[1]=v1;
}

void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const
key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], delta=0x9E3779B9, sum=delta*num_rounds;
```

```
    for (i=0; i < num_rounds; i++) {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) &
3]);
        sum -= delta;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
    }
    v[0]=v0; v[1]=v1;
}


uint32_t string_to_uint32(uint8_t *buff ,int length){
    uint32_t res, digit;
    int i;
    uint8_t partial;
    res = 0;

    for(i=0;i<length;i++){
        partial = buff[i];

        switch (partial) {
            case '1':
                digit = 1;
                break;
            case '2':
                digit = 2;
                break;
            case '3':
                digit = 3;
                break;
            case '4':
                digit = 4;
                break;
            case '5':
                digit = 5;
                break;
            case '6':
                digit = 6;
                break;
            case '7':
                digit = 7;
                break;
            case '8':
                digit = 8;
                break;
            case '9':
                digit = 9;
                break;
            case '0':
                digit = 0;
                break;

            default: // For character '\n'
                digit = 0;
                break;
        }
        res = res + digit * pow32(10,10-(i+1));

    }

    return res;
}
```

```
uint32_t pow32(int a, int b){
    int i;
    uint32_t res = (uint32_t)1;

    for(i=0;i<b;i++)
        res = res * a;
    return res;
}

int rearrange(uint8_t array[BUFF_SIZE]){ //Returns 0 if '\n' was found on
the string
    uint8_t extra[BUFF_SIZE];
    int i,cont;
    int ctrl=20;
    int res = 1;

    for(i=0;i<BUFF_SIZE;i++){
        if(array[i]=='\n'){
            ctrl = i;
            res = 0;
        }
    }

    cont=0;
    if(ctrl != 10){
        while(cont<BUFF_SIZE){
            if (++ctrl==BUFF_SIZE){
                ctrl = 0;
            }
            extra[cont]=array[ctrl];
            cont++;
        }

        for(i=0;i<BUFF_SIZE;i++)
            array[i] = extra[i];
    }

    return res;
}
```

A.3 Libraries header files

A.3.1 nfc.h

```
#ifndef NFC_H_
#define NFC_H_

#include <stdio.h>
#include <inttypes.h>
#include "drivers/mss_i2c/mss_i2c.h"
#include <drivers/mss_gpio/mss_gpio.h>

// Target address (got it from datasheet)
#define PN532_I2C_ADDRESS (0x48>>1)
#define PN532_I2C_ADDRESS_WRITE 0x48 //DONT USE
#define PN532_I2C_ADDRESS_READ 0x49 // DONT USE
```

```
// Initialization bytes
#define PN532_PREAMBLE (0x00)
#define PN532_STARTCODE2 (0x00FF)
#define PN532_HOST_TO_NFC (0xD4)
#define PN532_NFC_TO_HOST (0xD5)
#define PN532_POSTAMBLE (0x00)

//Commands
#define PN532_COMMAND_GETFIRMWARE (0x02)
#define PN532_COMMAND_POWERDOWN (0x16)
#define PN532_COMMAND_GETSTATUS (0x04)
#define PN532_COMMAND_INAUTOPOLL (0x60)
#define PN532_COMMAND_RF_CONFIG (0x32)
#define PN532_COMMAND_INLISTPASSIVETARGET (0x4A)
#define PN532_COMMAND_SAMCONFIG (0x14)

// Prototypes
uint8_t nfc_get_response_size();

uint8_t nfc_read_ack();

uint8_t nfc_read_response();

uint8_t *nfc_get_response_buffer();

void nfc_init(void);

void nfc_send_command(const uint8_t command, const uint8_t info_buf[],
const uint8_t length);

uint8_t nfc_GetGeneralStatus();

uint8_t nfc_InListPassiveTarget();

uint8_t nfc_SAMConfig();

uint8_t nfc_GetFirmwareVersion();

uint8_t nfc_is_interrupt_handled(void);

void nfc_set_interrupt_handled(uint8_t val);

uint8_t nfc_last_was_ack();
#endif /* NFC_H_ */
```

A.3.2 lora.h

```
#ifndef LORA_H_
#define LORA_H_

#include "drivers/mss_spi/drivers/mss_spi/mss_spi.h"
#include "drivers/mss_gpio/mss_gpio.h"
#include <inttypes.h>
#include <time.h>
#include "drivers/mss_uart/mss_uart.h"
#include "lock.h"

#ifdef LOCK_0
#define DEVICE_ID 0xD0
```

```
#endif

#ifdef LOCK_1
#define DEVICE_ID 0xD1
#endif

#define GATEWAY_ID 0x66


#define LORA_RegFifoTxBaseAddr 0x0E

// This is the maximum number of interrupts the driver can support
// Most Arduinos can handle 2, Megas can handle more
#define RH_RF95_NUM_INTERRUPTS 3

// Max number of octets the LORA Rx/Tx FIFO can hold
#define RH_RF95_FIFO_SIZE 255

// This is the maximum number of bytes that can be carried by the LORA.
// We use some for headers, keeping fewer for RadioHead messages
#define RH_RF95_MAX_PAYLOAD_LEN RH_RF95_FIFO_SIZE

// The length of the headers we add.
// The headers are inside the LORA's payload
#define RH_RF95_HEADER_LEN 4

// This is the maximum message length that can be supported by this
driver.
// Can be pre-defined to a smaller size (to save SRAM) prior to including
this header
// Here we allow for 1 byte message length, 4 bytes headers, user data
and 2 bytes of FCS
#ifndef RH_RF95_MAX_MESSAGE_LEN
 #define RH_RF95_MAX_MESSAGE_LEN (RH_RF95_MAX_PAYLOAD_LEN -
RH_RF95_HEADER_LEN)
#endif

// The crystal oscillator frequency of the module
#define RH_RF95_FXOSC 32000000.0

// The Frequency Synthesizer step = RH_RF95_FXOSC / 2^^19
#define RH_RF95_FSTEP  (RH_RF95_FXOSC / 524288)


// Register names (LoRa Mode, from table 85)
#define RH_RF95_REG_00_FIFO                                0x00
#define RH_RF95_REG_01_OP_MODE                             0x01
#define RH_RF95_REG_02_RESERVED                            0x02
#define RH_RF95_REG_03_RESERVED                            0x03
#define RH_RF95_REG_04_RESERVED                            0x04
#define RH_RF95_REG_05_RESERVED                            0x05
#define RH_RF95_REG_06_FRF_MSB                             0x06
#define RH_RF95_REG_07_FRF_MID                             0x07
#define RH_RF95_REG_08_FRF_LSB                             0x08
#define RH_RF95_REG_09_PA_CONFIG                           0x09
#define RH_RF95_REG_0A_PA_RAMP                             0x0a
#define RH_RF95_REG_0B_OCP                                 0x0b
#define RH_RF95_REG_0C_LNA                                 0x0c
#define RH_RF95_REG_0D_FIFO_ADDR_PTR                       0x0d
#define RH_RF95_REG_0E_FIFO_TX_BASE_ADDR                   0x0e
#define RH_RF95_REG_0F_FIFO_RX_BASE_ADDR                   0x0f
```

```
#define RH_RF95_REG_10_FIFO_RX_CURRENT_ADDR             0x10
#define RH_RF95_REG_11_IRQ_FLAGS_MASK                   0x11
#define RH_RF95_REG_12_IRQ_FLAGS                        0x12
#define RH_RF95_REG_13_RX_NB_BYTES                      0x13
#define RH_RF95_REG_14_RX_HEADER_CNT_VALUE_MSB          0x14
#define RH_RF95_REG_15_RX_HEADER_CNT_VALUE_LSB          0x15
#define RH_RF95_REG_16_RX_PACKET_CNT_VALUE_MSB          0x16
#define RH_RF95_REG_17_RX_PACKET_CNT_VALUE_LSB          0x17
#define RH_RF95_REG_18_MODEM_STAT                       0x18
#define RH_RF95_REG_19_PKT_SNR_VALUE                    0x19
#define RH_RF95_REG_1A_PKT_RSSI_VALUE                   0x1a
#define RH_RF95_REG_1B_RSSI_VALUE                       0x1b
#define RH_RF95_REG_1C_HOP_CHANNEL                      0x1c
#define RH_RF95_REG_1D_MODEM_CONFIG1                    0x1d
#define RH_RF95_REG_1E_MODEM_CONFIG2                    0x1e
#define RH_RF95_REG_1F_SYMB_TIMEOUT_LSB                 0x1f
#define RH_RF95_REG_20_PREAMBLE_MSB                     0x20
#define RH_RF95_REG_21_PREAMBLE_LSB                     0x21
#define RH_RF95_REG_22_PAYLOAD_LENGTH                   0x22
#define RH_RF95_REG_23_MAX_PAYLOAD_LENGTH               0x23
#define RH_RF95_REG_24_HOP_PERIOD                       0x24
#define RH_RF95_REG_25_FIFO_RX_BYTE_ADDR                0x25
#define RH_RF95_REG_26_MODEM_CONFIG3                    0x26

#define RH_RF95_REG_40_DIO_MAPPING1                     0x40
#define RH_RF95_REG_41_DIO_MAPPING2                     0x41
#define RH_RF95_REG_42_VERSION                          0x42

#define RH_RF95_REG_4B_TCXO                             0x4b
#define RH_RF95_REG_4D_PA_DAC                           0x4d
#define RH_RF95_REG_5B_FORMER_TEMP                      0x5b
#define RH_RF95_REG_61_AGC_REF                          0x61
#define RH_RF95_REG_62_AGC_THRESH1                      0x62
#define RH_RF95_REG_63_AGC_THRESH2                      0x63
#define RH_RF95_REG_64_AGC_THRESH3                      0x64

// RH_RF95_REG_01_OP_MODE                       0x01
#define RH_RF95_LONG_RANGE_MODE                 0x80
#define RH_RF95_ACCESS_SHARED_REG               0x40
#define RH_RF95_MODE                            0x07
#define RH_RF95_MODE_SLEEP                      0x00
#define RH_RF95_MODE_STDBY                      0x01
#define RH_RF95_MODE_FSTX                       0x02
#define RH_RF95_MODE_TX                         0x03
#define RH_RF95_MODE_FSRX                       0x04
#define RH_RF95_MODE_RXCONTINUOUS               0x05
#define RH_RF95_MODE_RXSINGLE                   0x06
#define RH_RF95_MODE_CAD                        0x07

// RH_RF95_REG_09_PA_CONFIG                     0x09
#define RH_RF95_PA_SELECT                       0x80
#define RH_RF95_MAX_POWER                       0x70
#define RH_RF95_OUTPUT_POWER                    0x0f

// RH_RF95_REG_0A_PA_RAMP                       0x0a
#define RH_RF95_LOW_PN_TX_PLL_OFF               0x10
#define RH_RF95_PA_RAMP                         0x0f
#define RH_RF95_PA_RAMP_3_4MS                   0x00
#define RH_RF95_PA_RAMP_2MS                     0x01
#define RH_RF95_PA_RAMP_1MS                     0x02
#define RH_RF95_PA_RAMP_500US                   0x03
```

```
#define RH_RF95_PA_RAMP_250US                0x0
#define RH_RF95_PA_RAMP_125US                0x05
#define RH_RF95_PA_RAMP_100US                0x06
#define RH_RF95_PA_RAMP_62US                 0x07
#define RH_RF95_PA_RAMP_50US                 0x08
#define RH_RF95_PA_RAMP_40US                 0x09
#define RH_RF95_PA_RAMP_31US                 0x0a
#define RH_RF95_PA_RAMP_25US                 0x0b
#define RH_RF95_PA_RAMP_20US                 0x0c
#define RH_RF95_PA_RAMP_15US                 0x0d
#define RH_RF95_PA_RAMP_12US                 0x0e
#define RH_RF95_PA_RAMP_10US                 0x0f

// RH_RF95_REG_0B_OCP                        0x0b
#define RH_RF95_OCP_ON                       0x20
#define RH_RF95_OCP_TRIM                     0x1f

// RH_RF95_REG_0C_LNA                        0x0c
#define RH_RF95_LNA_GAIN                     0xe0
#define RH_RF95_LNA_BOOST                    0x03
#define RH_RF95_LNA_BOOST_DEFAULT            0x00
#define RH_RF95_LNA_BOOST_150PC              0x11

// RH_RF95_REG_11_IRQ_FLAGS_MASK             0x11
#define RH_RF95_RX_TIMEOUT_MASK              0x80
#define RH_RF95_RX_DONE_MASK                 0x40
#define RH_RF95_PAYLOAD_CRC_ERROR_MASK       0x20
#define RH_RF95_VALID_HEADER_MASK            0x10
#define RH_RF95_TX_DONE_MASK                 0x08
#define RH_RF95_CAD_DONE_MASK                0x04
#define RH_RF95_FHSS_CHANGE_CHANNEL_MASK     0x02
#define RH_RF95_CAD_DETECTED_MASK            0x01

// RH_RF95_REG_12_IRQ_FLAGS                  0x12
#define RH_RF95_RX_TIMEOUT                   0x80
#define RH_RF95_RX_DONE                      0x40
#define RH_RF95_PAYLOAD_CRC_ERROR            0x20
#define RH_RF95_VALID_HEADER                 0x10
#define RH_RF95_TX_DONE                      0x08
#define RH_RF95_CAD_DONE                     0x04
#define RH_RF95_FHSS_CHANGE_CHANNEL          0x02
#define RH_RF95_CAD_DETECTED                 0x01

// RH_RF95_REG_18_MODEM_STAT                 0x18
#define RH_RF95_RX_CODING_RATE               0xe0
#define RH_RF95_MODEM_STATUS_CLEAR           0x10
#define RH_RF95_MODEM_STATUS_HEADER_INFO_VALID 0x08
#define RH_RF95_MODEM_STATUS_RX_ONGOING      0x04
#define RH_RF95_MODEM_STATUS_SIGNAL_SYNCHRONIZED 0x02
#define RH_RF95_MODEM_STATUS_SIGNAL_DETECTED 0x01

// RH_RF95_REG_1C_HOP_CHANNEL                0x1c
#define RH_RF95_PLL_TIMEOUT                  0x80
#define RH_RF95_RX_PAYLOAD_CRC_IS_ON         0x40
#define RH_RF95_FHSS_PRESENT_CHANNEL         0x3f

// RH_RF95_REG_1D_MODEM_CONFIG1              0x1d
#define RH_RF95_BW                           0xc0
#define RH_RF95_BW_125KHZ                    0x00
#define RH_RF95_BW_250KHZ                    0x40
#define RH_RF95_BW_500KHZ                    0x80
```

```
#define RH_RF95_BW_RESERVED                         0xc0
#define RH_RF95_CODING_RATE                         0x38
#define RH_RF95_CODING_RATE_4_5                     0x00
#define RH_RF95_CODING_RATE_4_6                     0x08
#define RH_RF95_CODING_RATE_4_7                     0x10
#define RH_RF95_CODING_RATE_4_8                     0x18
#define RH_RF95_IMPLICIT_HEADER_MODE_ON             0x04
#define RH_RF95_RX_PAYLOAD_CRC_ON                   0x02
#define RH_RF95_LOW_DATA_RATE_OPTIMIZE              0x01

// RH_RF95_REG_1E_MODEM_CONFIG2                     0x1e
#define RH_RF95_SPREADING_FACTOR                    0xf0
#define RH_RF95_SPREADING_FACTOR_64CPS              0x60
#define RH_RF95_SPREADING_FACTOR_128CPS             0x70
#define RH_RF95_SPREADING_FACTOR_256CPS             0x80
#define RH_RF95_SPREADING_FACTOR_512CPS             0x90
#define RH_RF95_SPREADING_FACTOR_1024CPS            0xa0
#define RH_RF95_SPREADING_FACTOR_2048CPS            0xb0
#define RH_RF95_SPREADING_FACTOR_4096CPS            0xc0
#define RH_RF95_TX_CONTINUOUS_MOE                   0x08
#define RH_RF95_AGC_AUTO_ON                         0x04
#define RH_RF95_SYM_TIMEOUT_MSB                     0x03

// RH_RF95_REG_4D_PA_DAC                            0x4d
#define RH_RF95_PA_DAC_DISABLE                      0x04
#define RH_RF95_PA_DAC_ENABLE                       0x07

#define RH_BROADCAST_ADDRESS 0xff

#define LORA_MODE_IDLE 0
#define LORA_MODE_SLEEP 1
#define LORA_MODE_RX 2
#define LORA_MODE_TX 3
#define LORA_MODE_CAD 4

#define FALSE 0
#define TRUE 1

uint8_t LORA_init(void);
void LORA_handle_interrupt(void);
void LORA_set_mode_idle(void);
void LORA_sleep(void);
void LORA_set_mode_tx(void);
void LORA_set_mode_rx(void);
void LORA_set_modem_config(uint8_t *config);
void LORA_set_preamble_length(uint16_t bytes);
void LORA_set_frequency(double f);
void LORA_set_tx_power(int8_t power, uint8_t useRFO);
void LORA_validate_rx_buf(void);
uint8_t LORA_available(void);
void LORA_clear_rx_buf(void);

void LORA_wait_available(void);

uint8_t LORA_wait_available_timeout(uint16_t timeout);

uint8_t LORA_wait_packet_sent(uint16_t timeout);

void LORA_set_promiscuous(uint8_t prom);

void LORA_set_this_address(uint8_t addr);
```

```
void LORA_set_header_to(uint8_t to);

void LORA_set_header_from(uint8_t from);

void LORA_set_header_id(uint8_t id);

void LORA_set_header_flags(uint8_t set, uint8_t clear);

uint8_t LORA_send(const uint8_t* data, uint8_t len);
uint8_t LORA_recv(uint8_t* buf, uint8_t* len);


uint8_t LORA_read(uint8_t addr);
void LORA_write(uint8_t addr, uint8_t data);

uint8_t LORA_burst_read(uint8_t addr, uint8_t* res, uint8_t len);
uint8_t LORA_burst_write(uint8_t addr, uint8_t* src, uint8_t len);


#endif /* LORA_H_ */
```

A.3.3 servo.h

```
#ifndef SERVO_H_
#define SERVO_H_

#include <stdint.h>

#define SERVO_ADDR 0x40050004
#define SERVO_PERIOD 500000
#define SERVO_UNLOCKED 23000
#define SERVO_LOCKED 45500 // 37500

#define SERVO ((uint32_t *) SERVO_ADDR)

void SERVO_init(void);

void SERVO_lock(void);

void SERVO_unlock(void);

uint32_t SERVO_read_state(void);

#endif /* SERVO_H_ */
```

A.3.4 neopixel.h

```
#ifndef NEOPIXEL_H_
#define NEOPIXEL_H_

#include <stdint.h>

#define NP_ADDR 0x40050000
#define NP_CONTROL_SET 0x01000000
#define NP_CONTROL_APPLY 0x02000000
#define NP_CONTROL_CLEAR 0x04000000

#define NP_COLOUR_BLUE 0x00200088
```

```
#define NP_COLOUR_YELLOW 0x0083A400
#define NP_COLOUR_GREEN 0x00901030
#define NP_COLOUR_RED 0x00209925

void NP_init(void);

void NP_set_pixel(uint32_t colour);

void NP_set_pixel_c(uint8_t red, uint8_t green, uint8_t blue);

void NP_apply(void);

void NP_clear(void);

uint32_t NP_get_pixel(void);

#endif /* NEOPIXEL_H_ */
```

A.3.5 contact_switch.h

```
#ifndef CONTACT_SWITCH_H_
#define CONTACT_SWITCH_H_

#include <stdint.h>
#include <drivers/mss_gpio/mss_gpio.h>

#define CS_DOOR_OPEN 1
#define CS_DOOR_CLOSED 0

void CS_init(void);

uint8_t CS_get_door_status(void);

#endif /* CONTACT_SWITCH_H_ */
```

A.3.6 rsa.h

```
#ifndef RSA_H_
#define RSA_H_

#include <stdint.h>

#define RSA_BIT_SWITCH_ADDR 0x40050008
#define RSA_MESSAGE_ADDR 0x4005000C
#define RSA_MODULUS_ADDR 0x40050010
#define RSA_EXPONENT_ADDR 0x40050014
#define RSA_RESIDUE_ADDR 0x40050018
#define RSA_BEGIN_ENCRYPT_ADDR 0x4005001C
#define RSA_RESULT_ADDR 0x40050020
#define RSA_RESULT_VALID_ADDR 0x40050024

#define LOAD_UPPER 0x1
#define LOAD_LOWER 0x0

#define RSA_BIT_SWITCH ((uint32_t *) RSA_BIT_SWITCH_ADDR)
#define RSA_MESSAGE ((uint32_t *) RSA_MESSAGE_ADDR)
#define RSA_MODULUS ((uint32_t *) RSA_MODULUS_ADDR)
#define RSA_EXPONENT ((uint32_t *) RSA_EXPONENT_ADDR)
#define RSA_RESIDUE ((uint32_t *) RSA_RESIDUE_ADDR)
```

```
#define RSA_BEGIN_ENCRYPT ((uint32_t *) RSA_BEGIN_ENCRYPT_ADDR)
#define RSA_RESULT ((uint32_t *) RSA_RESULT_ADDR)
#define RSA_RESULT_VALID ((uint32_t *) RSA_RESULT_VALID_ADDR)

void RSA_init(void);

void RSA_run(uint32_t* message, uint32_t *key, uint32_t *modulus,
uint32_t *residue);

uint8_t RSA_read_result(uint32_t *result);

uint8_t RSA_is_ready(void);

#endif /* RSA_H_ */
```

A.3.7 RPi_FR.h

```
#ifndef RPi_FR_h
#define RPi_FR_h

#include "RPi_FR.h"
#include <inttypes.h>
#include "drivers/mss_uart/mss_uart.h"
#include <drivers/mss_gpio/mss_gpio.h>
#define ROUNDS 64
#define REC_OPEN_DOOR 3
#define REC_DONT_OPEN 1
#define KEY_OPEN 1250640089
#define KEY_LOCK 3530294326
#define BUFF_SIZE 11

void RPi_init(void);

int FaceRecognition(void);

void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const
key[4]);

void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const
key[4]);

uint32_t string_to_uint32(uint8_t *buff ,int length);

uint32_t pow32(int a, int b);

int rearrange(uint8_t array[BUFF_SIZE]);

#endif /* RPi_FR_h */
```

A.4 Verilog code

A.4.1 apb3_interface.v

```
// apb3_interface.v
module apb3_interface(
```

```
                        PCLK,
                        PENABLE,
                        PSEL,
                        PRESETN,
                        PWRITE,
                        PREADY,
                        PSLVERR,
                        PADDR,
                        PWDATA,
                        PRDATA,
                        NP_OUT,
                    SERVO_OUT,
                    FABINT,
                    RAM_DINA,
                    RAM_DINB,
                    RAM_ADDRA,
                    RAM_ADDRB,
                    RAM_RWA,
                    RAM_RWB,
                    RAM_BLKA,
                    RAM_BLKB,
                    RAM_RESET,
                    RAM_DOUTA,
                    RAM_DOUTB
);

// APB Bus Interface
input PCLK,PENABLE, PSEL, PRESETN, PWRITE;
input [31:0] PWDATA;
input [7:0] PADDR;
output [31:0] PRDATA;
output PREADY, PSLVERR;
output NP_OUT;
output SERVO_OUT;
//input PIN_NFC_IRQ;
output FABINT;
//output RSA_VALID;
//output TEST;
output [31:0] RAM_DINA;
output [31:0] RAM_DINB;
output [6:0] RAM_ADDRA;
output [6:0] RAM_ADDRB;
output RAM_RWA;
output RAM_RWB;
output RAM_BLKA;
output RAM_BLKB;
output RAM_RESET;
input [31:0] RAM_DOUTA;
input [31:0] RAM_DOUTB;

// Test Interface
// output [4:0] TPS; // Use for your debugging

wire [31:0] NP_PRDATA;
wire [31:0] SERVO_PRDATA;
wire [31:0] RSA_PRDATA;


assign BUS_WRITE_EN = (PENABLE & PWRITE & PSEL);
assign BUS_READ_EN = (~PWRITE & PSEL); //Data is ready during first cycle
to make it availble on the bus when PENABLE is asserted
```

```
assign PREADY = 1'b1;
assign PSLVERR = 1'b0;

wire NP_EN;
wire SERVO_EN;
wire RSA_EN;

//wire RSA_VALID;

assign NP_EN = ~(| PADDR[7:0]);
assign SERVO_EN = PADDR[7:0] == 8'h04;
//RSA has many internal addresses. Look at RSA.v to see assignments.
assign RSA_EN = (PADDR[7:0] == 8'h08 || PADDR[7:0] == 8'h0C
                || PADDR[7:0] == 8'h10 || PADDR[7:0] == 8'h14
                || PADDR[7:0] == 8'h18 || PADDR[7:0] == 8'h1C
                || PADDR[7:0] == 8'h20 || PADDR[7:0] == 8'h24);


assign PRDATA = (NP_EN ? NP_PRDATA :
                (SERVO_EN ? SERVO_PRDATA :
                (RSA_EN ? RSA_PRDATA :
                (32'b0))));

//assign PRDATA = RSA_PRDATA;


neopixel pxl_0(     .pclk(PCLK),
                    .nreset(PRESETN),
                    .bus_write_en(BUS_WRITE_EN),
                    //.bus_read_en(BUS_READ_EN),
                    .bus_addr(PADDR),
                    .bus_write_data(PWDATA),
                    //.bus_read_data(NP_PRDATA),
                   .np_en(NP_EN),
                   .np_out(NP_OUT)
                   //.green_t(TEST)
              );

servo servo_0(      .pclk(PCLK),
                    .nreset(PRESETN),
                    .bus_write_en(BUS_WRITE_EN),
                    //.bus_read_en(BUS_READ_EN),
                    .bus_addr(PADDR),
                    .bus_write_data(PWDATA),
                    //.bus_read_data(SERVO_PRDATA),
                   .servo_en(SERVO_EN),
                   .servo_out(SERVO_OUT)
              );

/*
nfc nfc(
                   .pclk(PCLK),
                    .nreset(PRESETN),
                    .bus_write_en(BUS_WRITE_EN),
                    .bus_read_en(BUS_READ_EN),
                    .bus_addr(PADDR),
                    .bus_write_data(PWDATA),
                    .bus_read_data(PRDATA),
                   .fabint(FABINT),
                   .irq_pin(PIN_NFC_IRQ)
```

```
          );
*/

rsa rsa_0(
                .pclk(PCLK), // clock
                .nreset(PRESETN), // system reset
                .bus_write_en(BUS_WRITE_EN),
                .bus_read_en(BUS_READ_EN),
                .RSA_ENABLE(RSA_EN),
                .bus_addr(PADDR), // I/O address
                .bus_write_data(PWDATA), // data from processor to I/O
device (32 bits)
                .bus_read_data(RSA_PRDATA), // data to processor from I/O
device (32-bits)
                .result_valid(FABINT),
                .RAM_DINA(RAM_DINA),
                .RAM_DINB(RAM_DINB),
                .RAM_ADDRA(RAM_ADDRA),
                .RAM_ADDRB(RAM_ADDRB),
                .RAM_RWA(RAM_RWA),
                .RAM_RWB(RAM_RWB),
                .RAM_BLKA(RAM_BLKA),
                .RAM_BLKB(RAM_BLKB),
                .RAM_DOUTA(RAM_DOUTA),
                .RAM_DOUTB(RAM_DOUTB)
            );

endmodule
```

A.4.2 MonMult.v

```
module MonMult(
    input pclk,
    input nreset,
    input GO,
    input [63:0] A,
    input [63:0] B,
    input [63:0] M,
    output reg [65:0] P,
    output reg is_ready);

    reg [65:0] P_n;
    reg is_ready_n;

    reg [6:0] counter;
    reg [6:0] counter_n;

    reg [2:0] state;
    reg [2:0] state_n;

    reg [63:0] add;
    reg [63:0] add_n;

    always @* begin
        P_n = P;
        is_ready_n = is_ready;
        counter_n = counter;
        state_n = state;
        add_n = add;
```

```
        if (~counter[6]) begin
            is_ready_n = 1'b0;
            //counter_n = counter + 1'b1;

            if(state == 3'd0) begin
                add_n = (P[0] ^ (A[counter] & B[0])) ? M : 64'b0;
                state_n = 3'd1;
            end else if(state == 3'd1) begin
                state_n = 3'd2;
            end else if(state == 3'd2) begin
                add_n = A[counter] ? B : 64'b0;
                state_n = 3'd3;
            end else if(state == 3'd3) begin
                state_n = 3'd4;
            end else if(state == 3'd4) begin
                P_n = P >> 1;
                state_n = 3'd5;
            end else if(state == 3'd5) begin
                add_n = ~M + 1;
                state_n = 3'd6;
            end else if(state == 3'd6) begin
                counter_n = counter + 1'b1;
                state_n = 3'd0;
            end

            if(state == 3'd1 || state == 3'd3 || (state == 3'd6 && (&
counter[5:0]) && P >= M) ) begin
                P_n = P + add;
            end

            //if(state == 3'd4 && (& counter[5:0])) begin
                //if( P_n >= M ) begin
                    //P_n = P_n - {2'b0, M};
                //end
            //end
        end


        else if (counter[6]) begin
            is_ready_n = 1'b1;
            counter_n = counter;
        end

    end

    always @(posedge pclk) begin
        if(~nreset | ~GO) begin
            P <= 66'b0;
            is_ready <= 1'b0;
            counter <= 7'b0;
            state <= 1'b0;
        end else begin
            counter <= counter_n;
            is_ready <= is_ready_n;
            P <= P_n;
            state <= state_n;
            add <= add_n;
        end
    end

endmodule
```

A.4.3 NFC.v

```
module nfc(
             input pclk,
             input nreset,
             input bus_write_en,
             input bus_read_en,
             input bus_addr,
             input bus_write_data,
             input bus_read_data,
            output fabint,
            input irq_pin
            );

    reg fabint;

    always @(posedge pclk) begin
        if (~nreset && (irq_pin == 0))
            fabint <= 1;
        else
            fabint <= 0;
    end

endmodule
```

A.4.4 RSA.v

```
`define KEY_LENGTH 64 //This is insecure... but this is just a proof of
concept.
`define BIT_SWITCH_ADDR 8'h08
`define MESSAGE_ADDR 8'h0C
`define MODULUS_ADDR 8'h10
`define EXPONENT_ADDR 8'h14
`define RESIDUE_ADDR 8'h18
`define RSA_ENCRYPT_ADDR 8'h1C
`define RESULT_ADDR 8'h20
`define RESULT_VALID_ADDR 8'h24
`define TEMP_RESULT_ADDR 8'h28

`define RAM_READ 1'b1
`define RAM_WRITE 1'b0

module rsa(
    input pclk, // clock
    input nreset, // system reset
    input bus_write_en,
    input bus_read_en,
    input RSA_ENABLE,
    input [7:0] bus_addr, // I/O address
    input wire [31:0] bus_write_data, // data from processor to I/O
device (32 bits)
    output reg [31:0] bus_read_data, // data to processor from I/O device
(32-bits)
    output reg result_valid,
    output reg [31:0] RAM_DINA,
    output reg [31:0] RAM_DINB,
    output reg [6:0] RAM_ADDRA,
    output reg [6:0] RAM_ADDRB,
```

```
    output reg RAM_RWA,
    output reg RAM_RWB,
    output reg RAM_BLKA,
    output reg RAM_BLKB,
    input [31:0] RAM_DOUTA,
    input [31:0] RAM_DOUTB
);

//////////////////////////////////////////////////////////////////////
////////////////////////////////
//
// This is used to minimize internal addresses.
// When set to 0, a r/w operation will assume lower 32bits of register.
// When set to 1, will assume upper 32bits.
// i.e. A write to address 8'h0C when bit_switch is 1 will set
message_upper = bus_write_data.
//
//////////////////////////////////////////////////////////////////////
////////////////////////////////
reg bit_switch = 0;
reg bit_switch_n;

reg [31:0] RAM_DINA_n;
reg [31:0] RAM_DINB_n;
reg [6:0] RAM_ADDRA_n;
reg [6:0] RAM_ADDRB_n;
reg RAM_RWA_n;
reg RAM_RWB_n;
reg RAM_BLKA_n;
reg RAM_BLKB_n;
reg [2:0] read_wait_counter;
reg [2:0] read_wait_counter_n;

reg [31:0] modulus_upper; //64-bit (8-byte) modulous (the important
part).
reg [31:0] modulus_lower;
reg [31:0] modulus_upper_n;
reg [31:0] modulus_lower_n;
/*
reg [31:0] message_upper; //message must be at most as large as the
modulus.
reg [31:0] message_lower;
reg [31:0] message_upper_n;
reg [31:0] message_lower_n;

reg [31:0] exponent_upper; //64-bit (8-byte) exponent (65,537 is nearly
universally the public exponenet, but the private exponent is normally
choosen to be close to the size of the modulus).
reg [31:0] exponent_lower;
reg [31:0] exponent_upper_n;
reg [31:0] exponent_lower_n;

reg [31:0] residue_upper; //This needs to be equal to R^2 mod M, where R
= 2^KEY_LENGTH and M=modulus.
reg [31:0] residue_lower;
reg [31:0] residue_upper_n;
reg [31:0] residue_lower_n;

reg [31:0] result_upper;
reg [31:0] result_lower;
reg [31:0] result_upper_n;
```

```
reg [31:0] result_lower_n;

reg [31:0] temp_result_upper;
reg [31:0] temp_result_lower;
reg [31:0] temp_result_upper_n;
reg [31:0] temp_result_lower_n;

reg [31:0] Z_upper;
reg [31:0] Z_lower;
reg [31:0] Z_upper_n;
reg [31:0] Z_lower_n;
*/

reg rsa_encrypt = 0; //When this is set to 1, the module will begin
encryption/decryption.
reg rsa_encrypt_n;

reg [4:0] encrypt_state;
reg [4:0] encrypt_state_n;
reg [6:0] counter;
reg [6:0] counter_n;

reg result_valid_n;

reg MonMult_GO;
reg MonMult_GO_n;
reg [63:0] A;
reg [63:0] A_n;
reg [63:0] B;
reg [63:0] B_n;
wire [65:0] P;
wire MonMult_ready;

MonMult MonMult_0(
    .pclk(pclk),
    .nreset(nreset),
    .GO(MonMult_GO),
    .A(A),
    .B(B),
    .M({modulus_upper, modulus_lower}),
    .P(P),
    .is_ready(MonMult_ready));


wire rsa_write;
wire rsa_read;
assign rsa_write = bus_write_en && RSA_ENABLE;
assign rsa_read = bus_read_en && RSA_ENABLE;

always @* begin
    bit_switch_n = bit_switch;
    modulus_upper_n = modulus_upper;
    modulus_lower_n = modulus_lower;
    rsa_encrypt_n = rsa_encrypt;
    A_n = A;
    B_n = B;
    encrypt_state_n = encrypt_state;
    counter_n = counter;
    result_valid_n = (encrypt_state == 5'd21);
    RAM_DINA_n = RAM_DINA;//32'b0;
    RAM_DINB_n = RAM_DINB;//32'b0;
```

```
RAM_ADDRA_n = RAM_ADDRA;//7'b0;
RAM_ADDRB_n = RAM_ADDRB;//7'b0;
RAM_RWA_n = RAM_RWA;//1'b0;
RAM_RWB_n = RAM_RWB;//1'b0;
RAM_BLKA_n = RAM_BLKA;//1'b1;
RAM_BLKB_n = RAM_BLKB;//1'b1;
read_wait_counter_n = read_wait_counter;
MonMult_GO_n = MonMult_GO;

bus_read_data = 32'bx;

if(bus_addr == `MODULUS_ADDR && rsa_write) begin
    if(bit_switch)
        modulus_upper_n = bus_write_data;
    else
        modulus_lower_n = bus_write_data;
end

if(bus_addr == `BIT_SWITCH_ADDR & rsa_write) begin
    bit_switch_n = bus_write_data[0];
end else if((bus_addr == `MESSAGE_ADDR ||
             bus_addr == `MODULUS_ADDR ||
             bus_addr == `EXPONENT_ADDR ||
             bus_addr == `RESIDUE_ADDR)
            && rsa_write) begin
    if(bit_switch) begin
        RAM_DINB_n = bus_write_data;
        RAM_BLKB_n = 1'b0;
        RAM_ADDRB_n = bus_addr[6:0] + 1;
        RAM_RWB_n = `RAM_WRITE;
    end else begin
        RAM_DINA_n = bus_write_data;
        RAM_BLKA_n = 1'b0;
        RAM_ADDRA_n = bus_addr[6:0];
        RAM_RWA_n = `RAM_WRITE;
    end
    rsa_encrypt_n = 1'b0;
    result_valid_n = 1'b0;
end else if(bus_addr == `RSA_ENCRYPT_ADDR) begin
    if (rsa_write) begin
        rsa_encrypt_n = bus_write_data[0];
        encrypt_state_n = 5'b0;
        if(rsa_encrypt == 1'b1 && bus_write_data[0] == 0) begin
            result_valid_n = 1'b0;
        end
    end
    if (rsa_read) begin
        bus_read_data = {31'b0, rsa_encrypt};
    end
end else if (bus_addr == `RESULT_ADDR & rsa_read) begin
        // preserve output of final monmult and return that
        bus_read_data = (bit_switch) ? P[63:32] : P[31:0];
end else if (bus_addr == `RESULT_VALID_ADDR & rsa_read) begin
        bus_read_data = {31'b0, result_valid};
end


//Encrypt FSM
if(rsa_encrypt) begin
    if(encrypt_state == 5'd0) begin
        // read residue in to A_n
```

```
        RAM_ADDRA_n = `RESIDUE_ADDR;
        RAM_ADDRB_n = `RESIDUE_ADDR + 1;
        RAM_RWA_n = `RAM_READ;
        RAM_RWB_n = `RAM_READ;
        RAM_BLKA_n = 1'b0;
        RAM_BLKB_n = 1'b0;
        MonMult_GO_n = 1'b0;
        read_wait_counter_n = 3'd0;
        //A_n = {residue_upper, residue_lower};
        B_n = 64'd1;
        encrypt_state_n = 5'd1;
    end else if(encrypt_state == 5'd1) begin
        if(read_wait_counter == 3'd5) begin
            A_n = {RAM_DOUTB, RAM_DOUTA};
            MonMult_GO_n = 1'b1;
            encrypt_state_n = 5'd2;
        end else begin
            read_wait_counter_n = read_wait_counter + 1;
        end
    end else if(encrypt_state == 5'd2) begin
        //MonMult_GO_n = 1'b1;
        if(MonMult_ready) begin
            encrypt_state_n = 5'd3;
        end
    end else if(encrypt_state == 5'd3) begin
        RAM_ADDRA_n = `RESULT_ADDR;
        RAM_ADDRB_n = `RESULT_ADDR + 1;
        RAM_RWA_n = `RAM_WRITE;
        RAM_RWB_n = `RAM_WRITE;
        RAM_BLKA_n = 1'b0;
        RAM_BLKB_n = 1'b0;
        RAM_DINA_n = P[31:0];
        RAM_DINB_n = P[63:32];
        MonMult_GO_n = 1'b0;
        encrypt_state_n = 5'd4;
    end else if(encrypt_state == 5'd4) begin
        // read message in to B
        RAM_ADDRA_n = `MESSAGE_ADDR;
        RAM_ADDRB_n = `MESSAGE_ADDR + 1;
        RAM_RWA_n = `RAM_READ;
        RAM_RWB_n = `RAM_READ;
        RAM_BLKA_n = 1'b0;
        RAM_BLKB_n = 1'b0;
        read_wait_counter_n = 3'd0;
        encrypt_state_n = 5'd5;
    end else if(encrypt_state == 5'd5) begin
        if(read_wait_counter == 3'd5) begin
            B_n = {RAM_DOUTB, RAM_DOUTA};
            MonMult_GO_n = 1'b1;
            encrypt_state_n = 5'd6;
        end else begin
            read_wait_counter_n = read_wait_counter + 1;
        end
    end else if(encrypt_state == 5'd6) begin
        if(MonMult_ready) begin
            encrypt_state_n = 5'd7;
        end
    end else if(encrypt_state == 5'd7) begin
        MonMult_GO_n = 1'b0;
        // read result in to A
        RAM_ADDRA_n = `RESULT_ADDR;
```

```
        RAM_ADDRB_n = `RESULT_ADDR + 1;
        RAM_RWA_n = `RAM_READ;
        RAM_RWB_n = `RAM_READ;
        RAM_BLKA_n = 1'b0;
        RAM_BLKB_n = 1'b0;
        //A_n = {result_upper, result_lower};
        B_n = P[63:0];
        read_wait_counter_n = 3'd0;
        counter_n = 1'b0;
        encrypt_state_n = 5'd8;
    end else if(encrypt_state == 5'd8) begin
        if(read_wait_counter == 3'd5) begin
            A_n = {RAM_DOUTB, RAM_DOUTA};
            MonMult_GO_n = 1'b1;
            encrypt_state_n = 5'd9;
        end else begin
            read_wait_counter_n = read_wait_counter + 1;
        end
    end else if(encrypt_state == 5'd9) begin
        if(MonMult_ready) begin
            encrypt_state_n = 5'd10;
        end
    end else if(encrypt_state == 5'd10) begin
        RAM_ADDRA_n = `TEMP_RESULT_ADDR;
        RAM_ADDRB_n = `TEMP_RESULT_ADDR + 1;
        RAM_RWA_n = `RAM_WRITE;
        RAM_RWB_n = `RAM_WRITE;
        RAM_BLKA_n = 1'b0;
        RAM_BLKB_n = 1'b0;
        RAM_DINA_n = P[31:0];
        RAM_DINB_n = P[63:32];
        MonMult_GO_n = 1'b0;
        A_n = B;
        encrypt_state_n = 5'd11;
    end else if(encrypt_state == 5'd11) begin
        MonMult_GO_n = 1'b1;
        if(MonMult_ready & MonMult_GO) begin
            encrypt_state_n = 5'd12;
        end
    end else if(encrypt_state == 5'd12) begin
        B_n = P[63:0];
        MonMult_GO_n = 1'b0;
        RAM_ADDRA_n = `EXPONENT_ADDR;
        RAM_ADDRB_n = `EXPONENT_ADDR + 1;
        RAM_RWA_n = `RAM_READ;
        RAM_RWB_n = `RAM_READ;
        RAM_BLKA_n = 1'b0;
        RAM_BLKB_n = 1'b0;
        read_wait_counter_n = 3'd0;
        encrypt_state_n = 5'd13;
    end else if(encrypt_state == 5'd13) begin
        if(read_wait_counter == 3'd5) begin
            encrypt_state_n = 5'd14;
        end else begin
            read_wait_counter_n = read_wait_counter + 1;
        end
    end else if(encrypt_state == 5'd14) begin
        counter_n = counter + 1'b1;
        if((counter[5] & RAM_DOUTB[counter[4:0]]) |
           (~counter[5] & RAM_DOUTA[counter[4:0]])) begin
            RAM_ADDRA_n = `TEMP_RESULT_ADDR;
```

```
            RAM_ADDRB_n = `TEMP_RESULT_ADDR + 1;
            RAM_RWA_n = `RAM_READ;
            RAM_RWB_n = `RAM_READ;
            RAM_BLKA_n = 1'b0;
            RAM_BLKB_n = 1'b0;
            read_wait_counter_n = 3'd0;
            encrypt_state_n = 5'd15;
        end else begin
            encrypt_state_n = 5'd17;
        end
    end else if(encrypt_state == 5'd15) begin
        if(read_wait_counter == 3'd5) begin
            RAM_DINA_n = RAM_DOUTA;
            RAM_DINB_n = RAM_DOUTB;
            encrypt_state_n = 5'd16;
        end else begin
            read_wait_counter_n = read_wait_counter + 1;
        end
    end else if(encrypt_state == 5'd16) begin
        RAM_ADDRA_n = `RESULT_ADDR;
        RAM_ADDRB_n = `RESULT_ADDR + 1;
        RAM_RWA_n = `RAM_WRITE;
        RAM_RWB_n = `RAM_WRITE;
        RAM_BLKA_n = 1'b0;
        RAM_BLKB_n = 1'b0;
        //RAM_DINA_n = RAM_DOUTA;
        //RAM_DINB_n = RAM_DOUTB;
        encrypt_state_n = 5'd17;
    end else if(encrypt_state == 5'd17) begin
        RAM_ADDRA_n = `RESULT_ADDR;
        RAM_ADDRB_n = `RESULT_ADDR + 1;
        RAM_RWA_n = `RAM_READ;
        RAM_RWB_n = `RAM_READ;
        RAM_BLKA_n = 1'b0;
        RAM_BLKB_n = 1'b0;
        read_wait_counter_n = 3'd0;
        encrypt_state_n = 5'd18;
    end else if(encrypt_state == 5'd18) begin
        if(read_wait_counter == 3'd5) begin
            encrypt_state_n = 5'd19;
        end else begin
            read_wait_counter_n = read_wait_counter + 1;
        end
    end else if(encrypt_state == 5'd19) begin
        if(counter[6]) begin
            A_n = 64'd1;
            B_n = {RAM_DOUTB, RAM_DOUTA};
            encrypt_state_n = 5'd20;
            MonMult_GO_n = 1'b1;
        end else begin
            A_n = {RAM_DOUTB, RAM_DOUTA};
            //B_n is always set to Z;
            encrypt_state_n = 5'd9;
            MonMult_GO_n = 1'b1;
        end
    end else if(encrypt_state == 5'd20) begin
        if(MonMult_ready) begin
            encrypt_state_n = 5'd21;
        end
    end else if(encrypt_state == 5'd21) begin
        //
```

```
            //result_upper_n = P[63:32];
            //result_lower_n = P[31:0];
            encrypt_state_n = 5'd21;
        end
    end
end

always @(posedge pclk) begin
    if(!nreset) begin
        bit_switch <= 1'b0;
        //message_upper <= 32'b0;
        //message_lower <= 32'b0;
        modulus_upper <= 32'b0;
        modulus_lower <= 32'b0;
        //exponent_upper <= 32'b0;
        //exponent_lower <= 32'b0;
        //residue_upper <= 32'b0;
        //residue_lower <= 32'b0;
        rsa_encrypt <= 1'b0;
        //result_upper <= 32'b0;
        //result_lower <= 32'b0;
        //temp_result_upper <= 32'b0;
        //temp_result_lower <= 32'b0;
        //Z_upper <= 32'b0;
        //Z_lower <= 32'b0;
        result_valid <= 1'b0;
        MonMult_GO <= 1'b0;
        A <= 64'b0;
        B <= 64'b0;
        encrypt_state <= 5'b0;
        counter <= 7'b0;
    end else begin
        RAM_DINA <= RAM_DINA_n;
        RAM_DINB <= RAM_DINB_n;
        RAM_ADDRA <= RAM_ADDRA_n;
        RAM_ADDRB <= RAM_ADDRB_n;
        RAM_RWA <= RAM_RWA_n;
        RAM_RWB <= RAM_RWB_n;
        RAM_BLKA <= RAM_BLKA_n;
        RAM_BLKB <= RAM_BLKB_n;
        bit_switch <= bit_switch_n;
        read_wait_counter <= read_wait_counter_n;
        //message_upper <= message_upper_n;
        //message_lower <= message_lower_n;
        modulus_upper <= modulus_upper_n;
        modulus_lower <= modulus_lower_n;
        //exponent_upper <= exponent_upper_n;
        //exponent_lower <= exponent_lower_n;
        //residue_upper <= residue_upper_n;
        //residue_lower <= residue_lower_n;
        rsa_encrypt <= rsa_encrypt_n;
        //result_upper <= result_upper_n;
        //result_lower <= result_lower_n;
        //temp_result_upper <= temp_result_upper_n;
        //temp_result_lower <= temp_result_lower_n;
        //Z_upper <= Z_upper_n;
        //Z_lower <= Z_lower_n;
        result_valid <= result_valid_n;
        MonMult_GO <= MonMult_GO_n;
        A <= A_n;
        B <= B_n;
```

```
        encrypt_state <= encrypt_state_n;
        counter <= counter_n;
    end
end


endmodule
```

A.4.5 servo.v

```
// servo.v
`define SERVO_PERIOD 500000

module servo(
           /*** APB3 BUS INTERFACE ***/
           input             pclk, // clock
           input             nreset, // system reset
           input                bus_write_en,
            //input               bus_read_en,
            input servo_en,
 //          input             PWRITE, // distinguishes read and write
cycles
           input [7:0]      bus_addr, // I/O address
           input wire [31:0] bus_write_data, // data from processor to
I/O device (32 bits)
           //output reg [31:0] bus_read_data, // data to processor from
I/O device (32-bits)
           output reg        servo_out
           );

   wire                      write_pulse;

   assign write_pulse = bus_write_en & servo_en;
   //assign read_pulse = bus_read_en & servo_en;

   reg [31:0]                pulse_comp;
   reg [31:0]                pulse_comp_n;

   reg [31:0]                counter;
   reg [31:0]                counter_n;

   reg servo_out_n;

   always @* begin
      counter_n = counter + 1'b1;
      pulse_comp_n = pulse_comp;
      servo_out_n = servo_out;
      //bus_read_data = 32'b0;

      if(write_pulse) begin
         pulse_comp_n = bus_write_data;
      end // if (write_pulse)

/*
      if (read_pulse) begin
        bus_read_data = pulse_comp;
      end // if (read_pulse)
*/

      if (counter < pulse_comp) begin
```

```
        servo_out_n = 1'b1;
      end else if (counter < `SERVO_PERIOD) begin
        servo_out_n = 1'b0;
      end else if (counter == `SERVO_PERIOD) begin
        servo_out_n = 1'b0;
        counter_n = 32'b0;
      end // if (counter < pulse_comp)
   end // always @*

   always @(posedge pclk) begin
      if (!nreset) begin
         pulse_comp <= 32'b0;
         counter <= 32'b0;
         servo_out <= 1'b0;
      end else begin
         pulse_comp <= pulse_comp_n;
         counter <= counter_n;
         servo_out <= servo_out_n;
      end // if (!nreset)
   end // always @(posedge pclk)

endmodule
```

A.4.6 neopixel.v

```
// neopixel.v
// clk period = 40 ns
`define CLOCK_PERIOD 40

`define PERIOD 31 // / `CLOCK_PERIOD // 1.25us = 1250 ns = 125
`define HOLD 1250 // 50us
`define PERIOD_PLUS_HOLD `PERIOD + `HOLD

`define HIGH_0 7'd10// 8'd32, 0.4us
`define HIGH_1 7'd20 // 8'd64, 0.8us

`define LOW_0 7'd21 // 0.85us
`define LOW_1 7'd11 // 0.45us

module neopixel(
           /*** APB3 BUS INTERFACE ***/
           input             pclk, // clock
           input             nreset, // system reset
           input                bus_write_en,
            //input               bus_read_en,
            input np_en,
 //          input             PWRITE, // distinguishes read and write
cycles
           input [7:0]      bus_addr, // I/O address
           input wire [31:0] bus_write_data, // data from processor to
I/O device (32 bits)
           //output reg [31:0] bus_read_data, // data to processor from
I/O device (32-bits)
           output reg        np_out
            //output [23:0] pixel_state
           );

   wire                      write_pulse;
   wire read_pulse;
```

```
   assign write_pulse = bus_write_en & np_en;
   //assign read_pulse = bus_read_en & np_en;

   reg [23:0]                neopixel_reg;
   reg [23:0]                neopixel_reg_n;
   reg [4:0]                 send_pixel;
   reg [4:0]                 send_pixel_n;

   reg [13:0]                counter;
   reg [13:0]                counter_n;
   //reg [13:0]                compare_reg;
   //reg [13:0]                compare_reg_n;

   reg np_out_n;

   //assign pixel_state = neopixel_reg;

   always @* begin
      counter_n = counter;
      neopixel_reg_n = neopixel_reg;
      np_out_n = np_out;
      //compare_reg_n = compare_reg;
      send_pixel_n = send_pixel;
      //bus_read_data = 32'b0;

      if (send_pixel != 5'b0) begin
         counter_n = counter + 1;
            /*
         if (neopixel_reg[23]) begin
            compare_reg_n = high_one;
         end else begin
            compare_reg_n = high_zero;
         end // if (neopixel_reg[23])
            */

         if ((neopixel_reg[23] && counter < `HIGH_1) ||
             (~neopixel_reg[23] && counter < `HIGH_0)) begin
            np_out_n = 1'b1;
         end else if (counter < `PERIOD) begin
            np_out_n = 1'b0;
         end else if (counter == `PERIOD && send_pixel != 25) begin
            neopixel_reg_n[23:1] = neopixel_reg[22:0];
            neopixel_reg_n[0] = neopixel_reg[23];
            send_pixel_n = send_pixel + 1;
            counter_n = 14'b0;
         end // if (counter < compare_reg)

         if (send_pixel == 25 && counter < `PERIOD_PLUS_HOLD) begin
            np_out_n = 1'b0;
         end else if (send_pixel == 25 && counter == `PERIOD_PLUS_HOLD)
begin
            send_pixel_n = 1'b0;
         end // if (send_pixel == 25 && counter < `PERIOD_PLUS_HOLD)

      end else if(write_pulse) begin // ignore write if currently sending
         if (bus_write_data[24]) begin // sets register
            neopixel_reg_n = bus_write_data[23:0];
         end else if (bus_write_data[25]) begin // begins send
            send_pixel_n = 5'b1;
         end else if (bus_write_data[26]) begin // clears register
            neopixel_reg_n = 24'b0;
```

```
         end // if (bus_write_data[24])
      end // if (send_pixel != 0)

/*
      if (read_pulse) begin
        bus_read_data = {8'b0, neopixel_reg};
      end // if (read_pulse)
*/

      if (send_pixel == 5'b0) begin
         counter_n = 8'b0;
      end // if (send_pixel == 5'b0)
   end // always @*

   always @(posedge pclk) begin
      if (!nreset) begin
         neopixel_reg <= 24'b0;
         send_pixel <= 8'b0;
         counter <= 8'b0;
         //compare_reg <= 8'b0;
         np_out <= 1'b0;
      end else begin
         neopixel_reg <= neopixel_reg_n;
         send_pixel <= send_pixel_n;
         counter <= counter_n;
         //compare_reg <= compare_reg_n;
         np_out <= np_out_n;
      end // if (!nreset)
   end // always @(posedge pclk)

endmodule
```

A.5 Python code

A.5.1 server.py (Lora gateway)

```
1.  #/usr/bin/python2
2.
3.  from bottle import Bottle, run, get, post, request, template, redirect,
    response
4.  from time import sleep
5.  from hashlib import sha256 as password_hash
6.  import sys
7.  import os
8.  import random
9.  import string
10. from keys import *
11.
12. sys.path.insert(0, '/usr/lib/python2.7/bridge')
13.
14. COOKIE_SECRET_KEY = "EECS373_LOCKNET_GO_BLUE"
15.
16. GRANT_ACCESS = 0xA6
17. DENY_ACCESS = 0xAD
18.
19. NUM_LOCKS = 2
20. LOCK_LIST = [0xD0, 0xD1]
21.
```

```
22. ON_WHITELIST = 0
23. ON_BLACKLIST = 1
24. ON_NEITHER = 2
25. UNASSOC = 3
26.
27.
28. LORA_REQ_ACCESS = 0xAC
29.
30. def encrypt(msg, lock):
31.     return pow(msg, PUBLIC_KEY, LOCK_MODULUS[lock])
32.
33. def decrypt(msg):
34.     return pow(msg, GATEWAY_PRI, GATEWAY_MODULUS)
35.
36. from bridgeclient import BridgeClient as bridgeclient
37. bridge = bridgeclient()
38.
39. locks = {}
40. for lock_id in LOCK_LIST:
41.     lock = {}
42.     lock['whitelist'] = set()
43.     lock['blacklist'] = set()
44.     lock['req_access'] = 0
45.
46.     fname = 'db/lock_{}.whitelist'.format(lock_id)
47.     if os.path.isfile(fname):
48.         with open(fname, 'r') as f:
49.             for line in f:
50.                 l = line.strip()
51.                 if l:
52.                     lock['whitelist'].add(l)
53.     else:
54.         open(fname, 'a').close()
55.
56.     fname = 'db/lock_{}.blacklist'.format(lock_id)
57.     if os.path.isfile(fname):
58.         with open(fname, 'r') as f:
59.             for line in f:
60.                 l = line.strip()
61.                 if l:
62.                     lock['blacklist'].add(l)
63.     else:
64.         open(fname, 'a').close()
65.
66.     locks[lock_id] = lock
67.
68. print(locks)
69.
70. def make_dict(keys, values):
71.     return dict(zip(keys, values + [None] * (len(keys) - len(values))))

72.
73. user_keys = ['first', 'last', 'username', 'password', 'pass_salt', 'nfc_
    id']
74. users = []
75. user_file = 'db/users.list'
76. if os.path.isfile(user_file):
77.     with open(user_file, 'r') as f:
78.         users = [make_dict(user_keys, line.strip().split(' ')) for line
    in f]
79. else:
```

```
80.     open(user_file, 'a').close()
81.
82. unassoc_tags = set()
83. fname = 'db/unassoc.tags'
84. if os.path.isfile(fname):
85.     with open(fname, 'r') as f:
86.         for line in f:
87.             l = line.strip()
88.             if l:
89.                 unassoc_tags.add(line.strip())
90. else:
91.     open(fname, 'a').close()
92.
93. def gen_salt():
94.     ALPHABET = string.ascii_letters + string.digits
95.     return ''.join(random.choice(ALPHABET) for i in range(15))
96.
97.
98. def check_lists(lock, nfc_tag):
99.     uid = ""
100.            print('checking for nfc_tag={}'.format(nfc_tag))
101.            for user in users:
102.                print('nfc_id={}'.format(user['nfc_id']))
103.                if (user['nfc_id'] == nfc_tag):
104.                    uid = user['username']
105.                    break
106.            print('user is {}'.format(uid))
107.            if not uid:
108.                return UNASSOC
109.            if uid in locks[lock]['whitelist']:
110.                return ON_WHITELIST
111.            if uid in locks[lock]['blacklist']:
112.                return ON_BLACKLIST
113.            return ON_NEITHER
114.
115.        def request_access(lock):
116.            locks[lock]['req_access'] = 1
117.
118.        def update_unassoc():
119.            print(unassoc_tags)
120.            with open('db/unassoc.locks', 'w') as f:
121.                for tag in unassoc_tags:
122.                    f.write('{}\n'.format(tag))
123.
124.        def update_db():
125.            print(locks)
126.            for i in locks.keys():
127.                lock = locks[i]
128.                with open('db/lock_{}.whitelist'.format(i), 'w') as f:
129.                    for uid in lock['whitelist']:
130.                        f.write('{}\n'.format(uid))
131.                with open('db/lock_{}.blacklist'.format(i), 'w') as f:
132.                    for uid in lock['blacklist']:
133.                        f.write('{}\n'.format(uid))
134.
135.        def update_users():
136.            with open('db/users.list', 'w') as f:
137.                for user in users:
138.                    f.write('{} {} {} {} {} {}\n'.format(user['first'], u
    ser['last'], user['username'], user['password'], user['pass_salt'], user
    ['nfc_id']))
```

```
139.
140.        def check_user_exists(username):
141.            return (username in [user['username'] for user in users])
142.
143.        def check_login(username, password):
144.            for user in users:
145.                if (user['username'] == username and
146.                    user['password'] == password_hash(password + user['pa
    ss_salt']).hexdigest()):
147.                    return True
148.            return False
149.
150.        def add_user(first, last, username, password, salt):
151.            if os.path.isfile(user_file):
152.                with open(user_file,'a') as f:
153.                    f.write("{} {} {} {} {}\n".format(first, last, userna
    me, password, salt))
154.
155.            users.append(make_dict(user_keys, [first, last, username, pas
    sword, salt]))
156.
157.        def add_to_list(lock, uid, l):
158.            n_l = 'whitelist' if l == 'blacklist' else 'blacklist'
159.            locks[lock][n_l].discard(uid)
160.            locks[lock][l].add(uid)
161.            update_db()
162.
163.        def remove_from_list(lock, uid, l):
164.            locks[lock][l].discard(uid)
165.            update_db()
166.
167.        def send_to_lora(lock, msg):
168.            bridge.mailbox(lock + '{0:#0{1}x}'.format(msg, 18))
169.
170.        app = Bottle()
171.
172.        @app.route('/assoc', method=['POST', 'GET'])
173.        def do_assoc():
174.            if request.method == 'POST':
175.                user = request.get_cookie("account", secret=COOKIE_SECRET
    _KEY)
176.                print('user in assoc is {}'.format(user))
177.                tag = request.forms.get('tag')
178.                for u in users:
179.                    print('checking against user {}'.format(u['username']
    ))
180.                    if u['username'] == user:
181.                        print('found user')
182.                        u['nfc_id'] = tag
183.                        unassoc_tags.discard(tag)
184.                        update_unassoc()
185.                        update_users()
186.                        break
187.                redirect('/')
188.            if request.method == 'GET':
189.                return template('assoc.tpl', user=request.get_cookie("acc
    ount", secret=COOKIE_SECRET_KEY), tags=unassoc_tags)
190.
191.
192.        @app.route('/enroll', method=['POST', 'GET'])
193.        def do_enroll():
```

```
194.             if request.method == 'POST':
195.                 username = request.forms.get('username')
196.                 if(check_user_exists(username)):
197.                     return "Enrollment Failed," + username + "is taken"
198.                 password = request.forms.get('password')
199.                 salt = gen_salt()
200.                 hashed = password_hash(password + salt).hexdigest()
201.                 name_first = request.forms.get('name_first')
202.                 name_last = request.forms.get('name_last')
203.                 add_user(name_first, name_last, username, hashed, salt)
204.                 response.set_cookie("account", username, secret=COOKIE_SE
    CRET_KEY)
205.                 redirect('/')
206.             if request.method == 'GET':
207.                 return template('enroll.tpl')
208. 
209.         @app.route('/login', method=['POST', 'GET'])
210.         def do_login():
211.             if request.method == 'POST':
212.                 username = request.forms.get('username')
213.                 password = request.forms.get('password')
214.                 if(check_login(username, password)):
215.                     response.set_cookie("account", username, secret=COOKI
    E_SECRET_KEY)
216.                     redirect('/')
217.                 else:
218.                     return "Login Failed"
219.             if request.method == 'GET':
220.                 return template('login.tpl')
221. 
222.         @app.route('/')
223.         def index():
224.             user = request.get_cookie("account", secret=COOKIE_SECRET_KEY
    )
225.             if not user:
226.                 redirect('/login')
227.             return template('index.tpl', user=user, locks=locks)
228. 
229.         @app.route('/grant', method='POST')
230.         def grant():
231.             lock = int(request.forms.get('lock'), 16)
232.             print('granting access to {}'.format(lock))
233.             locks[lock]['req_access'] = 0
234.             send_to_lora(hex(lock), encrypt(GRANT_ACCESS, lock))
235.             redirect('/')
236. 
237.         @app.route('/deny', method='POST')
238.         def deny():
239.             lock = int(request.forms.get('lock'), 16)
240.             locks[lock]['req_access'] = 0
241.             send_to_lora(hex(lock), encrypt(DENY_ACCESS, lock) )
242.             redirect('/')
243. 
244.         @app.route('/l')
245.         def from_lora():
246.             msg = int(request.query['m'], 16)
247.             lock = int(request.query['l'], 16)
248.             decr = decrypt(msg)
249.             print('recv: {}'.format(hex(msg)))
250.             print('decrypted: {}'.format(hex(decr)))
251.             resp = check_lists(lock, hex(decr))
```

```
252.             if resp == ON_WHITELIST:
253.                 print('granting access')
254.                 send_to_lora(hex(lock), encrypt(GRANT_ACCESS, lock))
255.             elif resp == ON_BLACKLIST:
256.                 print('denying access')
257.                 send_to_lora(hex(lock), encrypt(DENY_ACCESS, lock))
258.             elif resp == ON_NEITHER:
259.                 print('tag recognized, not on list')
260.                 request_access(lock)
261.             else:
262.                 print('unassociated tag')
263.                 unassoc_tags.add(hex(decr))
264.                 update_unassoc()
265.                 request_access(lock)
266.             return 'recv'
267.
268.         @app.route('/modify_list', method='POST')
269.         def modify_list():
270.             uid = request.forms.get('id')
271.             lock = int(request.forms.get('lock'), 16)
272.             l = request.forms.get('list')
273.             addrem = request.forms.get('addrem')
274.             if addrem == 'add':
275.                 add_to_list(lock, uid, l)
276.             elif addrem == 'rem':
277.                 remove_from_list(lock, uid, l)
278.             redirect('/')
279.
280.         run(app, host='0.0.0.0', port=8080, debug=True)
```

A.5.2 keys.py

```
1.  GATEWAY_MODULUS = 0xeda515ef24029417
2.  GATEWAY_PRI = 0x44de026cabdb311
3.  PUBLIC_KEY = 0x10001
4.
5.  LOCK_MODULUS = {}
6.  LOCK_MODULUS[0xD0] = 0xd8c0938e985082eb
7.  LOCK_MODULUS[0xD1] = 0xbc8686e283f127ff
```

A.5.3 main.py (Raspberry Pi)

```
1.  #!/usr/bin/env python
2.
3.  # Libraries that wil be used
4.  import recognition as rec
5.  import PIR
6.  import time
7.  import encryption as encr #RENOMBRAR ESTA LIBRARIA PARA QUE FUNCIONE
8.  import numpy as np
9.  import uart
10. from timer import Timer as tm
11. import serial
12. import email_lib as email
13. import memory
14.
15. # Constants
16. PIN_PIR = 17
```

```
SAMPLE_TIME = 10
NUM_ROUNDS_ENCR = 64
KEY = [np.uint32(0x34AE),np.uint32(0x44FB),np.uint32(0x19CD),np.uint32(0xAF19)]
OPEN_DOOR = 0x03
DONT_OPEN = 0x01
FIX_MSSG = 0xA3
SENDING_TIME = 0.5

# Steps of operation:
# 0) Initialization: storage data set
# 1) To be in idle mode with regular checks every 30 seconds
# 2) Check if there is a person with the PIR sensor
# 3) If there is someone, activate the the face recognition process
# 4) If the person is detected, generate a positive response, encrypt it and send it to SF
# 5) Otherwise, do the same with a negative response
# 6) Come back to the idling mode

def main():

    answer = raw_input("Do you want to include an additional face in data base?: ")
    while(answer in ['Yes','YES','yes','y','Y']):
        rec.RecognizerCreator()
        answer = raw_input("Do you want to include an additional face in data base?: ")
    print('Skipping initialization...')

    # Variable declaration
    phase = 1
    st = 1
    isDetected = 0
    pir = 0
    message = [np.uint32(0x00), np.uint32(0x00)]


    # PIR pin initialization
    PIR.init(PIN_PIR)

    # UART port initialization
        port = serial.Serial(
            port='/dev/serial0',
            baudrate=57600,
            parity=serial.PARITY_NONE,
            stopbits=serial.STOPBITS_ONE,
            bytesize=serial.EIGHTBITS,
            timeout=1
        )


    while(True):
        # Read the state of PIR sensor
        pir = PIR.read(PIN_PIR)
                print('Sampling time: ' + str(st))

        if(pir == 1): # Someone is in front of the door
            print('*** Body detected ***')
            print('Face detection phase '+str(phase)+' activated\n')
            isDetected = rec.Detector() # Activates detection for 30 seconds
```

```
73.             print('Face recognition ended. Detected: '+str(isDetected)+'
    \n')
74.             phase = phase + 1
75.
76.             if(isDetected):
77.                 # Generate positive message
78.                 message = [np.uint32(OPEN_DOOR), np.uint32(FIX_MSSG)]
79.
80.                 # Encrypt message
81.                 encr.encipher(NUM_ROUNDS_ENCR,message,KEY)
82.                 data_sent = str(message[0])+'\n'
83.
84.                 # Send message to the SF
85.                 timer = tm(saved_time=time.time(),diff_time = 0,limit_ti
    me = time.time() + SENDING_TIME)
86.                 port.write(data_sent)
87.
88.             else:
89.                 # Generate negative message
90.                 message = [np.uint32(DONT_OPEN), np.uint32(FIX_MSSG)]
91.                 #email.send_email(e_addr)
92.                                 names = []
93.                                 ids = []
94.                                 ems = []
95.                                 n_users = memory.read_users(names,ids,em
    s)
96.                                 for i in range(n_users):
97.                                     email.send_email(names[i],ems[i])
98.
99.                 # Encrypt message
100.                        encr.encipher(NUM_ROUNDS_ENCR,message,KEY)
101.                        data_sent = str(message[0])+'\n'
102.
103.                        # Send message to the SF
104.                        timer = tm(saved_time=time.time(),diff_time = 0,l
    imit_time = time.time() + SENDING_TIME)
105.                        port.write(data_sent)
106.
107.
108.
109.             #SFcomm.send(message)
110.             time.sleep(SAMPLE_TIME) # Sleep until next sample
111.
112.             st = st + 1
113.
114.             return
115.
116.
117.     if __name__ == '__main__':
118.         main()
```

A.5.4 comm.py (communication with SF)

```
1.  #!/usr/bin/env python
2.  import serial
3.
4.  class uart:
5.      def __init__(self):
6.          self.port = serial.Serial(
```

```
            port='/dev/serial0',
            baudrate=57600,
            parity=serial.PARITY_NONE,
                stopbits=serial.STOPBITS_ONE,
                bytesize=serial.EIGHTBITS,
                timeout=1
            )

    def send(message):
        self.port.write(message)
        return

    def read(buff):
        buff = self.port.readline()
        return

    def close():
        self.port.close()
```

A.5.5 email_lib.py

```
#!/usr/bin/env python
import smtplib
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
from email.MIMEBase import MIMEBase
from email import encoders

def data_email():
    print('** Email information **')
    toaddr = raw_input('Please introduce your email address: ')
    return toaddr

def send_email(name,toaddr):
    fromaddr = "dummyemailpy@gmail.com"
    msg = MIMEMultipart()

    msg['From'] = fromaddr
    msg['To'] = toaddr
    msg['Subject'] = "URGENT: UNKNOWN PERSON DETECTED"
    body = "Dear" + str(name)+",\nIt has been detected an unknown person trying to get into your home. Please find attached a picture of his/her face.\nYours sincerely,\nThe Security Team"

    msg.attach(MIMEText(body, 'plain'))

    filename = "image.png"
    attachment = open("image.jpg", "rb")

    part = MIMEBase('application', 'octet-stream')
    part.set_payload((attachment).read())
    encoders.encode_base64(part)
    part.add_header('Content-Disposition', "attachment; filename= %s" % filename)

    msg.attach(part)

    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.starttls()
```

```
    server.login(fromaddr, "asdfHJKL1")
    text = msg.as_string()
    server.sendmail(fromaddr, toaddr, text)
    server.quit()
```

A.5.6 encryption.py

```
import numpy as np
import time

def encipher(num_rounds,v,key):
  v0 = np.uint32(v[0])
  v1 = np.uint32(v[1])

  sums = np.uint32(0)
  delta = np.uint32(0x9E3779B9)

  for i in range(0,num_rounds):
    v0 += np.uint32((((v1 << 4) ^ (v1 >> 5)) + v1)) ^ np.uint32((sums + key[sums & 3]));
    sums += delta;
    v1 += np.uint32((((v0 << 4) ^ (v0 >> 5)) + v0)) ^ np.uint32((sums + key[(sums>>11) & 3]));


  v[0]=v0
  v[1]=v1

  return

def decipher(num_rounds,v,key):
  v0=v[0]
  v1=v[1]
  delta = np.uint32(0x9E3779B9)
  sums = np.uint32(delta*num_rounds)

  for i in range(0,num_rounds):

    v1 -= np.uint32((((v0 << 4) ^ (v0 >> 5)) + v0)) ^ np.uint32((sums + key[(sums>>11) & 3]));
    sums -= delta;
    v0 -= np.uint32((((v1 << 4) ^ (v1 >> 5)) + v1)) ^ np.uint32((sums + key[sums & 3]));


  v[0]=v0
  v[1]=v1

  return


def main():
  num_rounds = 64
  key = [np.uint32(0x34AE),np.uint32(0x44FB),np.uint32(0x19CD),np.uint32(0xAF19)]
  mes1 = [np.uint32(0x00), np.uint32(0xA3)]

```

```
  print('Initial message: ' + str(mes1[0]) + ' ' + str(mes1[1]) + '\n')

  encipher(num_rounds, mes1, key);

  print('Message encrypted: ' + str(mes1[0]) + ' ' + str(mes1[1]) + '\n'
)

  decipher(num_rounds, mes1, key);
  print('Message decrypted: ' + str(mes1[0]) + ' ' + str(mes1[1]) + '\n'
)



if __name__ == '__main__':
    main()
```

A.5.7 memory.py

```
#!/usr/bin/env python
# Library to work with the database of IDs used for Face recognition
# The file that will be modified should be named DataBase
# Format of the database:
# NAME  ID

# names, ids = read_users. Returns the number of users
def read_users(names,ids,ems):

    file = open("DataBase.txt","r")

    # Initializations
    lines = file.readlines()
    n_users = 0

    # Read
    for line in lines:

        names.append(line.split()[0])
        ids.append(line.split()[1])
        ems.append(line.split()[2])
        n_users = n_users + 1

    return n_users

def add_user(name, id, em):
    file = open("DataBase.txt","a")

    file.write(str(name) + ' ' + str(id) + ' ' + str(em) + '\n')
    file.close()

    return


def get_database():
    names = []
    ids = []
    ems = []
    n = read_users(names,ids,ems)
```

```
40.
41.     # Initialization of the dictionary that will be returned
42.     dic = {}
43.
44.     for i in range(n):
45.         # Warning, keys of the dictionary could be set as str
46.         dic[ids[i]] = names[i]
47.
48.     return dic
```

A.5.8 PIR.py

```
1.  #!/usr/bin/env python
2.
3.  """In this python file will be found the needed functions to interface (
    read) the PIR sensor"""
4.  import RPi.GPIO as GPIO
5.
6.  #Initialization
7.  def init(PIN_PIR):
8.      GPIO.setmode(GPIO.BCM)
9.      GPIO.setup(PIN_PIR,GPIO.IN)
10.
11. # Read: 1 if there is any IR detected, 0 ow
12. def read(PIN_PIR):
13.     output = GPIO.input(PIN_PIR)
14.     return output
```

A.5.9 recognition.py

```
1.  #!/usr/bin/env python
2.  import cv2,os
3.  import numpy as np
4.  from PIL import Image
5.  import memory
6.  import time
7.  from timer import Timer as tm
8.  WAITING_TIME = 30
9.  MIN_CORRELATION = 55
10.
11. """
12. Warning: the file "haarcascade_frontalface_default.xml" should be in the
     same execution directory to make it work.
13. The folders dataSet and trainner also need to be in the same path
14.
15.  Functions to use:
16.  RecognizerCreator
17.  Detector
18. """
19.
20. def RecognizerCreator():
21.     cam = cv2.VideoCapture(0)
22.     detector=cv2.CascadeClassifier('haarcascade_frontalface_default.xml'
    )
23.     print('Camera conected: '+ str(cam.isOpened()))
```

```
24.
25.     # Read data available to determine the id of the user
26.     names = []
27.     ids =[]
28.     ems = []
29.     current_Nusers = memory.read_users(names,ids,ems)
30.
31.     Id = current_Nusers + 1
32.
33.     Name = getName()
34.     em = raw_input('Introduce your email: ')
35.     # Add the new user to the DataBase
36.     memory.add_user(Name,Id,em)
37.
38.     n_photos = 0 # Lower bound of the number of pictures that will be us
    ed for the trainner
39.
40.     # Message for initalization (only displayed in terminal)
41.     print('Pictures will be taken in...')
42.     for i in [5,4,3,2,1]:
43.         print(str(i) +' seconds ...')
44.         time.sleep(1)
45.
46.
47.
48.     while(True):
49.         # Get image
50.         ret, img = cam.read()
51.
52.         # Image processing: obtain its gray color scale and obtain face
    characteristics
53.         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
54.         faces = detector.detectMultiScale(gray, 1.3, 5)
55.
56.         # Save and display face detected
57.         for (x,y,w,h) in faces:
58.
59.             # Establish frame in the face detected (will be shown in RPi
     desktop -> No headless mode)
60.             cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
61.
62.             # Increment the number of photos taken
63.             n_photos = n_photos + 1
64.
65.             # Save the image
66.             cv2.imwrite("dataSet/User."+ str(Id) +'.'+ str(n_photos) + "
    .jpg", gray[y:y+h,x:x+w])
67.
68.             cv2.imshow('frame',img)
69.
70.         # Loop control: get out of the loop if 'q' is pressed or if we r
    each the sample number
71.         # Wait for 100 miliseconds
72.         if cv2.waitKey(100) & 0xFF == ord('q'):
73.             break
74.
75.         # Number of samples that will be taken
76.         elif n_photos>20:
77.             break
78.
79.     # Check
```

```
80.     if not cam.isOpened():
81.         print('Camera could not be opened')
82.
83.
84.     # Clean objects
85.     cam.release()
86.     cv2.destroyAllWindows()
87.
88.     # Trainner creation
89.     recognizer = cv2.createLBPHFaceRecognizer()
90.     detector= cv2.CascadeClassifier("haarcascade_frontalface_default.xml
    ");
91.     faces,Ids = getImagesAndLabels('dataSet',detector)
92.     recognizer.train(faces, np.array(Ids))
93.
94.     # Save the reconizer that was created
95.     recognizer.save('trainner/trainner.yml')
96.     return
97.
98.
99. def getName():
100.           return raw_input('Please introduce new user name: ')
101.
102.       def getImagesAndLabels(path,detector):
103.
104.           # Get the path of the saved images
105.           imagePaths=[os.path.join(path,f) for f in os.listdir(path)]
106.
107.           # Initialize face list
108.           faceSamples=[]
109.
110.           # Initialize ID list
111.           Ids=[]
112.
113.           # Access to all image path, getting its Id and face images
114.           for imagePath in imagePaths:
115.
116.               # Load and convert to gray scale the image
117.               pilImage = Image.open(imagePath).convert('L')
118.
119.               # Convert PIL image into numpy array
120.               imageNp = np.array(pilImage,'uint8')
121.
122.               # Get the Id from the image
123.               Id = int(os.path.split(imagePath)[-1].split(".")[1])
124.
125.               # Face obtention
126.               faces = detector.detectMultiScale(imageNp)
127.
128.               # Append the face and ID to the empty lists that were cre
    ated before
129.               for (x,y,w,h) in faces:
130.                   faceSamples.append(imageNp[y:y+h,x:x+w])
131.                   Ids.append(Id)
132.
133.           return faceSamples,Ids
134.
135.
136.       def Detector():
137.           # Read Database
138.           DataBase = memory.get_database()
```

```
139.
140.            # Load recognizer
141.            recognizer = cv2.createLBPHFaceRecognizer()
142.            recognizer.load('trainner/trainner.yml')
143.            cascadePath = "haarcascade_frontalface_default.xml"
144.            faceCascade = cv2.CascadeClassifier(cascadePath);
145.
146.            # Initialize the camera. It will be opened for 1 minute
147.            timer = tm(saved_time=time.time(),diff_time = 0,limit_time =
    time.time() + WAITING_TIME)
148.            cam = cv2.VideoCapture(0)
149.            font = cv2.cv.InitFont(cv2.cv.CV_FONT_HERSHEY_SIMPLEX, 1, 1,
    0, 1, 1)
150.            cont = 0
151.            res = False
152.
153.            while (timer.diff_time + timer.saved_time < timer.limit_time)
    :
154.
155.                ret, im =cam.read()
156.                gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
157.                faces=faceCascade.detectMultiScale(gray, 1.2,5)
158.
159.                for(x,y,w,h) in faces:
160.
161.                    cv2.rectangle(im,(x,y),(x+w,y+h),(225,0,0),2)
162.                    Id, corr = recognizer.predict(gray[y:y+h,x:x+w])
163.
164.                    if (corr < MIN_CORRELATION):
165.                        text = DataBase[str(Id)]
166.                        cont = cont + 1
167.                    else:
168.                        text = "Unknown"
169.                        cont = 0
170.                    cv2.cv.PutText(cv2.cv.fromarray(im),text, (x,y+h),fon
    t, 255)
171.
172.                cv2.imshow('im',im)
173.
174.                if(cont>4):
175.                    res = True
176.                    break
177.
178.                if cv2.waitKey(10) and 0xFF==ord('q'):
179.                    break
180.
181.
182.                timer.update_diff_time()
183.
184.            # Save image to send later by email to the actual owner
185.            if(not res):
186.                cv2.imwrite("image.jpg", im)
187.
188.            cam.release()
189.            cv2.destroyAllWindows()
190.
191.            return res
```

A.5.10 timer.py

```
#!/usr/bin/env python
# Units in seconds
import time

class Timer:
    def __init__(self,saved_time, diff_time, limit_time):
        self.saved_time = saved_time
        self.diff_time = diff_time
        self.limit_time = limit_time
    # Methods -> Modify
    def save_time(self):
        self.saved_time = time.time()

    def update_diff_time(self):
        self.diff_time = time.time() - self.saved_time

    def set_limit_time(self,value):
        self.limit_time = time.time() + value

    # Methods -> Access
    def get_diff_time(self):
        return self.diff_time
```

A.5 Additional code

Here can be found templates for the webserver application and files with the key parameters for RSA encryption.

A.5.1 assoc.tpl

```
% include('header.tpl')
Associating tag for {{user}}
<form action="/assoc" method="POST">
Unassociated tags: <br>
% for tag in tags:
<input type="radio" name="tag" value="{{tag}}"> {{tag}}
% end
<input type="submit" value="Associate">
</form>
% include('footer.tpl')
```

A.5.2 enroll.tpl

```
% include('header.tpl')
<h2>Enroll</h2>
<form action="/enroll" method="POST">
    First Name: <input name="name_first" type="text"> <br>
    Last Name: <input name="name_last" type="text"> <br>
    Username: <input name="username" type="text"> <br>
    Password: <input name="password" type="password"> <br>
    <input value="Submit" type="submit" />
```

```
</form>
% include('footer.tpl')
```

A.5.3 header.tpl

```
<!doctype html>
<html>
<head>
  <title>LockNet</title>
  <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
</head>
<body>
```

A.5.4 index.tpl

```
% include('header.tpl')
Welcome, {{user}}! <br>

<a href="/assoc">Associate tag</a>
<h2>Blacklist</h2>
<form action="/modify_list" method="POST">
    Lock:
    % for id in locks.keys():
    <input name="lock" type="radio" value="{{hex(id)}}" {{"checked" if id == locks.keys()[0] else ""}}>{{hex(id)}}</input>
    % end
    <br>
    <input name="addrem" type="radio" value="add" checked>Add</input>
    <input name="addrem" type="radio" value="rem">Remove</input>
    <br>
    ID: <input name="id" type="text">
    <input name="list" type="hidden" value="blacklist">
    <input value="Submit" type="submit" />
</form>

<h2>Whitelist</h2>
<form action="/modify_list" method="POST">
    Lock:
    % for id in locks.keys():
    <input name="lock" type="radio" value="{{hex(id)}}" {{"checked" if id == locks.keys()[0] else ""}}>{{hex(id)}}</input>
    % end
    <br>
    <input name="addrem" type="radio" value="add" checked>Add</input>
    <input name="addrem" type="radio" value="rem">Remove</input>
    <br>
    ID: <input name="id" type="text">
    <input name="list" type="hidden" value="whitelist">
    <input value="Submit" type="submit" />
</form>

% for id in locks.keys():
    <div>
    <div>
        <h3>Lock {{hex(id)}} Whitelist members</h3>
        % for user in locks[id]['whitelist']:
```

```
40.         <p>{{user}}</p>
41.         % end
42.         </div>
43.
44.     <div>
45.         <h3>Lock {{hex(id)}} Blacklist members</h3>
46.         % for user in locks[id]['blacklist']:
47.         <p>{{user}}</p>
48.         % end
49.         </div>
50.     </div>
51. % end
52.
53. % include('req_access.tpl', lockslocks=locks)
54. % include('footer.tpl')
```

A.5.5 login.tpl

```
1.  % include('header.tpl')
2.  <h2>Login</h2>
3.  <form action="/login" method="POST">
4.      Username: <input name="username" type="text"> <br>
5.      Password: <input name="password" type="password"> <br>
6.      <input value="Submit" type="submit" />
7.  </form>
8.
9.  <a href="/enroll">Enroll</a>
10. % include('footer.tpl')
```

A.5.6 req_access.tpl

```
1.  % for id in locks.keys():
2.  LOCK {{hex(id)}}
3.  <form action="/grant" method="POST">
4.  <input name="lock" type="hidden" value="{{hex(id)}}">
5.  <input type="submit" value="GRANT ACCESS">
6.  </form>
7.  <form action="/deny" method="POST">
8.  <input name="lock" type="hidden" value="{{hex(id)}}">
9.  <input type="submit" value="DENY ACCESS">
10. </form>
11. % end
12. % end
```

A.5.7 RSA parameters

Gateway:

```
Private-Key: (64 bit)
modulus: 17124117274991694871 (0xeda515ef24029417)
publicExponent: 65537 (0x10001)
privateExponent: 310150406573241105 (0x44de026cabdb311)
prime1: 4188522047 (0xf9a7c63f)
prime2: 4088343593 (0xf3af2c29)
exponent1: 3610257135 (0xd73026ef)
exponent2: 2572393249 (0x99539b21)
```

```
coefficient: 2792533545 (0xa672ae29)

Public-Key: (64 bit)
Modulus: 17124117274991694871 (0xeda515ef24029417)
Exponent: 65537 (0x10001)
```

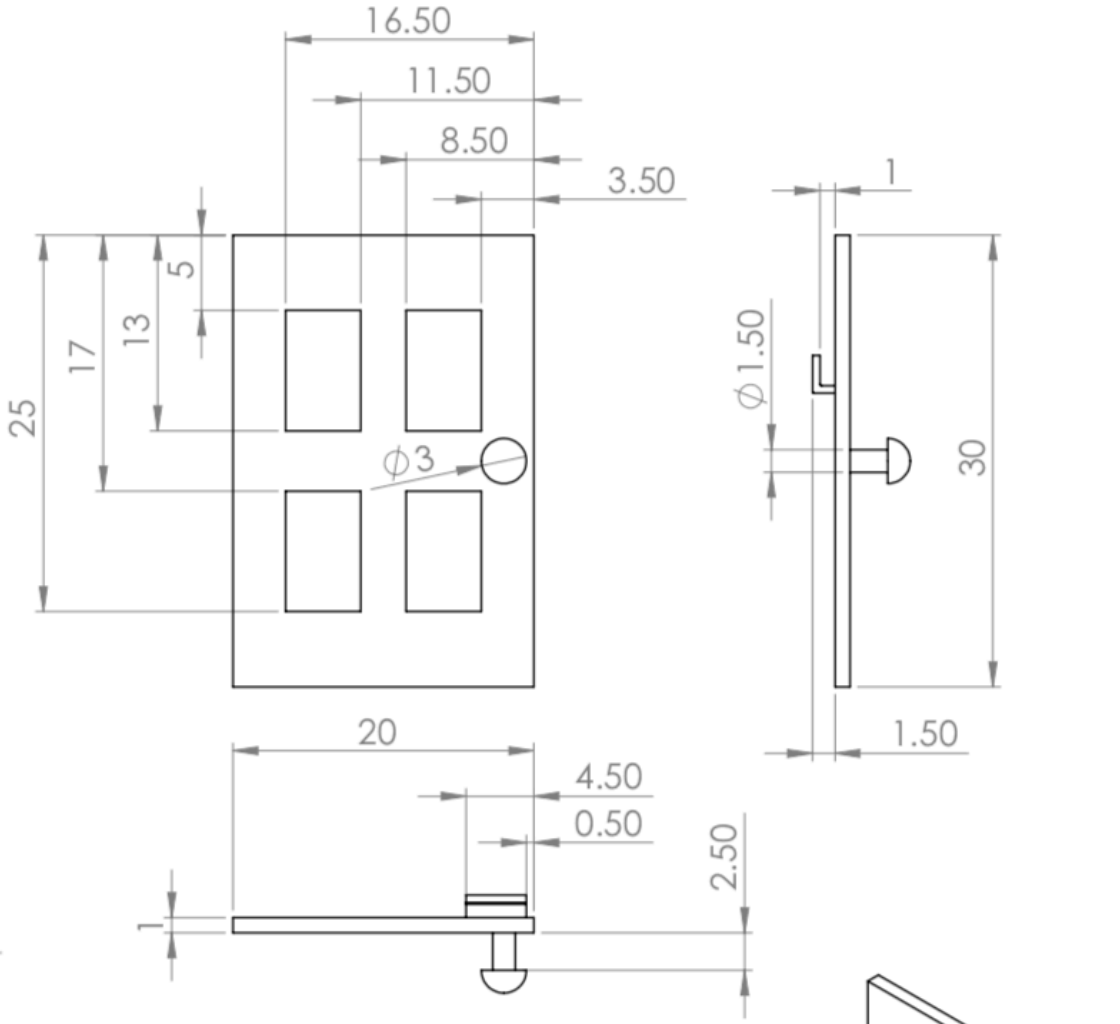
Lock controller:

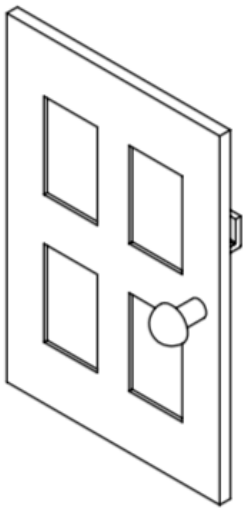
```
Private-Key: (64 bit)
modulus: 15618645748370932459 (0xd8c0938e985082eb)
publicExponent: 65537 (0x10001)
privateExponent: 5297570474157152241 (0x4984c0474cbc17f1)
prime1: 3958484633 (0xebf1ae99)
prime2: 3945612323 (0xeb2d4423)
exponent1: 2141690161 (0x7fa79931)
exponent2: 551110597 (0x20d947c5)
coefficient: 832263344 (0x319b54b0)

Public-Key: (64 bit)
Modulus: 15618645748370932459 (0xd8c0938e985082eb)
Exponent: 65537 (0x10001)
```

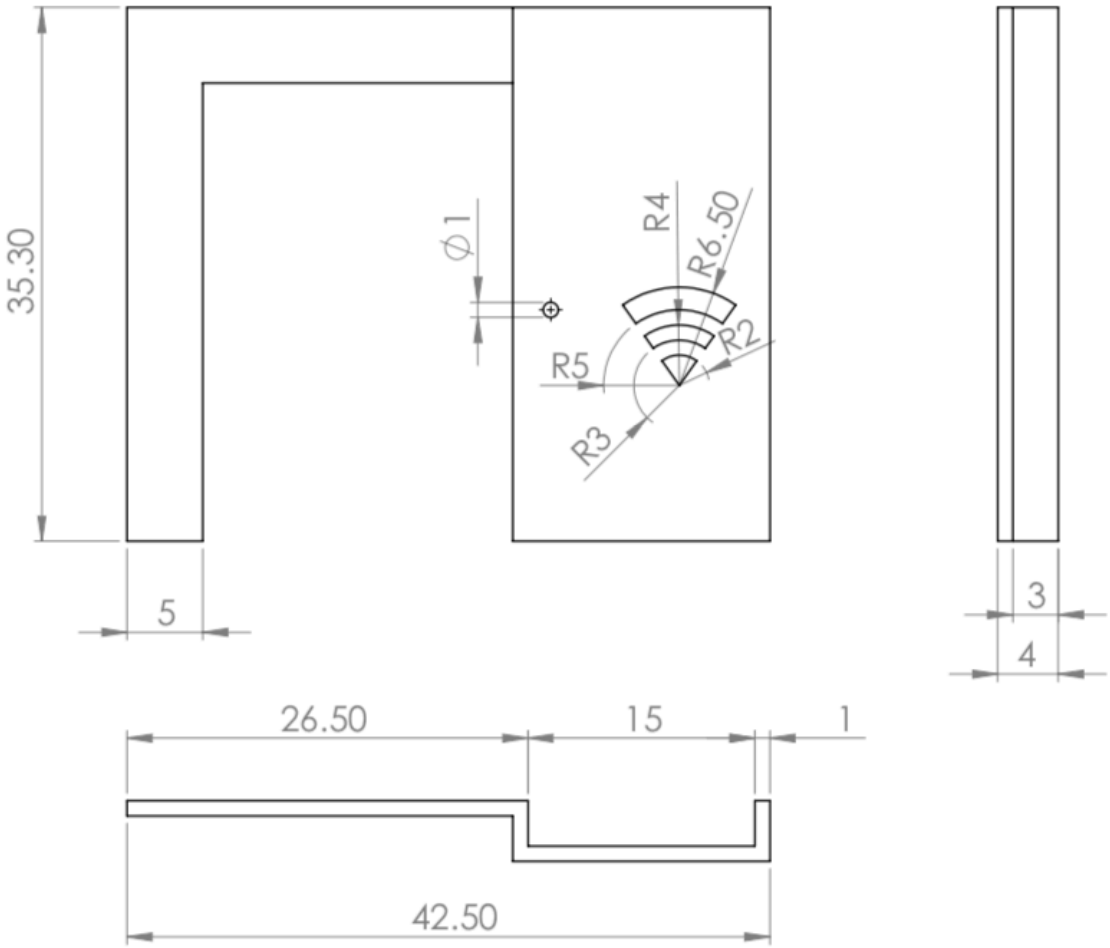
Appendix B

The design layout of the door and its frame are presented in Appendix B. These designs were made in SolidWorks and aimed to create a 3D printed little door to install the prototype and present it in the public exposition.





UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO. door	A4
				WEIGHT:		SCALE:1:5	SHEET 1 OF 1



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO. frame	A4
				WEIGHT:		SCALE:1:5	SHEET 1 OF 1

Appendix C

In Appendix C all the data gathered for face recognition tests is presented. These tests were carried out with 20 samples. The results are 0 or 1 (1 means the face was recognized, 0 if the algorithm detected it as unknown). Each iteration lasts 10 seconds and evaluates if the recognition was successful or not.

C.1 One subject with adjustable correlation test

Iterat.	Subject rec.
#1	0
#2	0
#3	0
#4	0
#5	0
#6	0
#7	0
#8	0
#9	0
#10	0

Table C.1: Recognition with correlation 10

Iterat.	Subject rec.
#1	0
#2	0
#3	0
#4	0
#5	0
#6	0
#7	0
#8	0
#9	0
#10	0

Table C.2: Recognition with correlation 15

Iterat.	Subject rec.
#1	1
#2	0
#3	0
#4	0
#5	0
#6	1
#7	0
#8	0
#9	1
#10	0

Table C.3: Recognition with correlation 20

Iterat.	Subject rec.
#1	1
#2	0
#3	1
#4	1
#5	1
#6	0
#7	1
#8	0
#9	0
#10	1

Table C.4: Recognition with correlation 25

Iterat.	Subject rec.
#1	1
#2	1
#3	1
#4	1
#5	1
#6	1
#7	1
#8	0
#9	1
#10	1

Table C.5: Recognition with correlation 30

Iterat.	Subject rec.
#1	1
#2	1
#3	1
#4	1
#5	1
#6	1
#7	1
#8	1
#9	1
#10	1

Table C.6: Recognition with correlation 35

C.2 One subject with adjustable samples test

This test evaluates how does the number of samples taken impact of the face recognition task. The correlation is fixed to 20 (based on the previous test it is the starting point to detect the face that was used to create the model).

Iterat.	Subject rec.
#1	0
#2	0
#3	0
#4	1
#5	0
#6	0
#7	0
#8	0
#9	0
#10	0

Table C.7: Recognition with 10 face samples

Iterat.	Subject rec.
#1	0
#2	1
#3	0
#4	0
#5	0
#6	0
#7	0
#8	1
#9	1
#10	0

Table C.8: Recognition with 20 face samples

Iterat.	Subject rec.
#1	1
#2	0
#3	1
#4	1
#5	0
#6	1
#7	0
#8	0
#9	1
#10	1

Table C.9: Recognition with 30 face samples

Iterat.	Subject rec.
#1	1
#2	0
#3	0
#4	0
#5	0
#6	0
#7	1
#8	0
#9	1
#10	1

Table C.10: Recognition with 40 face samples

Iterat.	Subject rec.
#1	1
#2	0
#3	1
#4	1
#5	0
#6	1
#7	0
#8	0
#9	1
#10	1

Table C.11: Recognition with 50 face samples

Iterat.	Subject rec.
#1	1
#2	0
#3	1
#4	1
#5	0
#6	0
#7	0
#8	1
#9	0
#10	1

Table C.12: Recognition with 60 face samples

C.3 Multiple subjects with adjustable correlation

This test focuses on multiple faces recognition. Adding a new user can produce a new type of error (when the program confuses one face with another). Therefore, there is a change in notation: 1 and 2 for the respective subjects and U for unknown.

Iteration	Subject #1	Subject #2
#1	1	U
#2	U	U
#3	1	U
#4	1	U
#5	1	U
#6	1	U
#7	1	U
#8	1	U
#9	1	2
#10	U	2

Table C.13: Multiple recognition with correlation 25

Iteration	Subject #1	Subject #2
#1	1	U
#2	1	U
#3	1	2
#4	1	U
#5	1	2
#6	U	2
#7	1	U
#8	1	U
#9	1	U
#10	U	2

Table C.14: Multiple recognition with correlation 30

Iteration	Subject #1	Subject #2
#1	1	2
#2	1	2
#3	1	2
#4	1	1
#5	1	1
#6	1	U
#7	1	U
#8	1	U
#9	1	1
#10	1	2

Table C.15: Multiple recognition with correlation 35

Iteration	Subject #1	Subject #1
#1	1	2
#2	1	2
#3	1	2
#4	2	1
#5	1	1
#6	1	1
#7	1	2
#8	1	2
#9	1	2
#10	1	2

Table C.16: Multiple recognition with correlation 40

Iteration	Subject #1	Subject #1
#1	1	2
#2	2	2
#3	1	2
#4	1	2
#5	2	1
#6	2	1
#7	1	2
#8	1	2
#9	1	2
#10	1	2

Table C.17: Multiple recognition with correlation 45

Iteration	Subject #1	Subject #1
#1	1	2
#2	1	2
#3	2	1
#4	1	2
#5	1	2
#6	2	2
#7	2	2
#8	1	1
#9	1	1
#10	2	2

Table C.18: Multiple recognition with correlation 50

Appendix D

All the datasheets of devices used in this project are presented in Appendix D. The links to the PDF files are provided in order to save space.

D.1 Datasheets

NFC:
https://www.nxp.com/docs/en/user-guide/141520.pdf

LoRa transceiver:
https://www.semtech.com/uploads/documents/sx1272.pdf

LoRa gateway:
http://www.dragino.com/downloads/downloads/UserManual/LG01_LoRa_Gateway_User_Manual.pdf

Neopixel LED:
https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf

PIR sensor:
https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf

SmartFusion Evaluation Kit:
https://www.eecs.umich.edu/courses/eecs373/labs/refs/A2F_EVAL_KIT_UG.pdf

ARM Cortex-M3 Embedded Software Development (AN-179): https://www.eecs.umich.edu/courses/eecs373/readings/ARM_Cortex_AppNote179.pdf

APB3 Protocol Specifications: https://www.eecs.umich.edu/courses/eecs373/labs/lab3/IHI0024C_amba_apb_protocol_spec.pdf

All these URLs were visited on 4th July 2018 with no problem.