ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

INTERCONEXIÓN DE DISPOSITIVOS PARA OBTENER UN REGISTRO DE CUMPLIMIENTO ACADÉMICO PARA LA ESFOT UTILIZANDO EL STM32F4

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO EN ELECTRÓNICA Y TELECOMUNICACIONES

ANGOS PICHO CÉSAR ANDRÉS
andresangosepn@gmail.com

GÓMEZ FARINANGO XIMENA MARISOL
ximena_epn@hotmail.com

DIRECTOR: ING. CARLOS ROMO
carlos_romo@epn.edu.ec

Quito, Junio, 2015

DECLARACIÓN

Nosotros, Ximena Marisol Gómez Farinango y César Andrés Angos Picho, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Ximena Marisol Gómez Farinango

César Andrés Angos Picho

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Ximena Marisol Gómez Farinango y César Andrés Angos Picho, bajo mi supervisión.

Ing. Carlos Romo

DIRECTOR DE PROYECTO

AGRADECIMIENTO

En primera instancia agradezco a mi padre celestial, por ser quien me levanta de cada tropiezo y guía cada uno de mis pasos.

A mis adorados padres Germancito y Delita, con quienes he podido contar siempre, por sus consejos, su amor, su motivación constante para alcanzar mis metas, por su sacrificio diario y entrega total con nosotros sus hijos.

A mis hermanos: Marco, Christian, Paola, Byron y Mireya por ser mis mejores amigos, gracias por todos los buenos momentos compartidos y por su apoyo incondicional.

A mi amado esposo Darwin, mi compañero de vida quien siempre ha estado ahí en los buenos y difíciles momentos y a toda su familia por apoyarme y brindarme su cariño.

Al Ing. Carlos Romo, por su acertada dirigencia para la culminación de este proyecto de titulación.

Y a todos quienes me han brindado su amistad sincera, sus conocimientos y su tiempo.

A todos ustedes mi mayor reconocimiento y gratitud.

Ximena Gómez

DEDICATORIA

Con mi más grande respeto e inmenso cariño dedico este proyecto:

A mi madre quien siempre creyó en mí, por apoyarme, por cuidarme en todo momento, por reír y llorar conmigo. Por todo ese inmenso amor demostrado que Dios la bendiga siempre.

A mi padre quien me ha enseñado a perseverar y no desmayar en los momentos más difíciles, por su arduo trabajo para ser de mí una persona de bien.

A mi esposo por brindarme su amor, su apoyo constante y comprensión.

A mis hermanos, familiares y amigos que con sus consejos y enseñanzas me han ayudado a crecer personal y profesionalmente.

Ximena Gómez

DEDICATORIA

Dedicado a mi familia, el amor de mi vida.

Andrés Angos

CONTENIDO

RESUMEN

Este proyecto de titulación tiene como finalidad crear un mecanismo para el registro de asistencia de los docentes que imparten clases los días sábados en la ESFOT mediante la construcción de un dispositivo electrónico que tiene la capacidad de registrar al docente mediante un sensor de huellas dactilares. El dispositivo también cuenta con una Tablet que se utiliza como interfaz final entre el dispositivo y el usuario permitiéndole al docente ingresar información referente al tema impartido en clase de manera amigable y segura. Toda esta información se podrá extraer mediante una interfaz USB. En vista de que el sistema mantiene un control de tiempo exacto se logra un registro eficaz del cumplimiento académico.

Capítulo 1. Expone los fundamentos teóricos de la tecnología de identificación de huellas dactilares, se describe también en forma general las características y clasificación de los dispositivos utilizados en la construcción del sistema.

Capítulo 2. Detalla las herramientas de software y hardware utilizado en la construcción del sistema, expone las características y requisitos específicos para el funcionamiento del dispositivo. Ofrece información referente a la estructuración del programa grabado en el microcontrolador, así como el que se ha desarrollado para la interfaz de usuario en la Tablet. Contiene pruebas y resultados del sistema además de su implementación.

Capítulo3. Presenta una guía que le ayuda al usuario a interactuar con el dispositivo, es decir ofrece información explícita y ordenada de como el usuario puede acceder a las diferentes funciones con las que cuenta el dispositivo, como son: proceso de ingreso de nuevos usuarios al sistema, proceso de identificación de usuarios, proceso de extracción de información, proceso de borrado de usuarios, etc.

Capítulo 4. Se presentan las conclusiones y recomendaciones del sistema.

CAPÍTULO 1

FUNDAMENTACIÓN TÉORICA

1.1 BIOMETRÍA [1]

Cada individuo cuenta con características morfológicas o rasgos físicos únicos que los diferencian del resto de los seres humanos.

El concepto etimológico de biometría se origina de las palabras bios que significa vida y metron que es medida, por lo tanto biometría se refiere al estudio matemático y estadístico de las características biológicas o físicas que permiten la identificación o verificación de un individuo basado en dichas características.

Actualmente los sistemas biométricos gozan de gran acogida debido a la evolución de los microprocesadores y componentes electrónicos que cuentan con una velocidad de procesamiento cada vez mayor, además de ser más compactos, esto permite automatizar la verificación de identidad en base a la construcción de un dispositivo sea factible, económica y eficaz para organizaciones públicas, privadas, militares y comerciales ya que los sistemas biométricos son actualmente considerados como el método ideal para identificación humana.

Estos sistemas están constituidos en base a software y hardware necesarios para reconocer parámetros fisiológicos del ser humano mediante sensores que acondicionan dichos parámetros como información con técnicas estadísticas y matemáticas para lograr una identificación satisfactoria del individuo.

Este sistema es capaz de:

- Obtener la muestra biométrica del individuo
- Extraer datos referentes a la muestra
- Comparar los datos obtenidos con los existentes en la base de datos
- Decidir la correspondencia de los datos
- Indicar el resultado de la verificación

1.1.1 PROCESOS DE AUTENTICACIÓN E IDENTIFICACIÓN BIOMÉTRICA

En el control en base a sistemas de análisis biométrico existe una fase de autenticación e identificación.

La categoría de autenticación (o verificación), proceso conocido como "uno-para-uno", consiste en obtener la imagen de la huella dactilar de un individuo, cuya identidad es conocida, para hacer una comparación con la imagen que está almacenada en la base de datos y verificar si la huella dactilar pertenece a ese individuo.

La categoría de identificación o "uno para muchos" consiste en reconocer solamente la imagen de la huella dactilar y compararla con las existentes en la base de datos para hallar la identidad del individuo al que pertenece esa huella dactilar.

1.1.2 FUNCIONAMIENTO Y RENDIMIENTO

Habitualmente en un sistema biométrico existe un proceso de registro, donde se adquieren características morfológicas del usuario las cuales se procesan mediante un algoritmo numérico para finalmente ser almacenados en la base de datos del sistema, esta información es única para cada individuo. En la autenticación e identificación del individuo las características biométricas son comparadas con las almacenadas en la base de datos y si concuerda permite el acceso al sistema. Los sistemas biométricos tienen tasas de error que varían desde 60% a 99.9%. [1]

Algunos de los parámetros para medir el rendimiento de un sistema biométrico son:

Tasa de Falsa Aceptación (False Acceptance Rate o FAR); es la probabilidad de que un individuo no registrado sea aceptado por equívoco.

En un sistema de verificación biométrica es posible obtener una medida de seguridad, según la ecuación: seguridad = (1 - FAR).

Tasa de Falso Rechazo (False Rejection Rate o FRR); es la probabilidad de que un usuario registrado sea rechazado.

[1] https://es.wikipedia.org/wiki/Biometría

A partir de la FRR es posible precisar la conveniencia de un sistema biométrico, de acuerdo a la ecuación: conveniencia = (1 - FRR).

Tasa de Error Igual (Equal Error Rate o EER); es un parámetro con el cual es posible igualar los factores FAR y FRR ya que son parámetros inversamente proporcionales. Esta medida es usada para determinar el umbral de calidad del sistema biométrico, cuando los errores de rechazo y aceptación son iguales el umbral del sistema (EER) se establece en 50%. Mientras menor valor tenga el EER el sistema es más fiable.

1.2 TIPOS DE SISTEMAS BIOMÉTRICOS [2]

La tecnología biométrica ha permitido el desarrollo de sistemas biométricos, basados en mediciones de características biológicas, psicológicas y sociales los cuales admiten llevar a cabo de manera automatizada la identificación y verificación de la identidad de los seres humanos. A continuación se describen las tecnologías biométricas con más presencia en el mercado.

1.2.1 RECONOCIMIENTO DE LA HUELLA DACTILAR

Basado en el principio de que no existen dos huellas dactilares iguales, el reconocimiento de huella dactilar es el sistema de identificación biométrica por excelencia debido a que es fácil de adquirir, eficaz y amigable con los usuarios.

Cuando un usuario desea autenticarse ante el sistema sitúa su dedo en un área de lectura. Aquí se toma una imagen que posteriormente se normaliza mediante un sistema de finos espejos para corregir ángulos, cada huella digital posee pequeños arcos, ángulos, bucles, remolinos es decir características únicas y es de esta imagen normalizada de la que el sistema extrae las minucias que va a comparar con las almacenadas en su base de datos.

Los sistemas basados en reconocimiento de huellas son económicos en comparación a otros biométricos, como los basados en el reconocimiento del iris. Sin embargo, tienen como desventaja la incapacidad temporal de autenticar usuarios que se hayan podido herir en el dedo a reconocer (un pequeño corte o una quemadura que afecte a varias minucias pueden hacer inútil al sistema). También elementos como la suciedad del dedo, la presión ejercida sobre el lector o el estado de la piel pueden ocasionar lecturas erróneas.

Figura 1.1: Reconocimiento de la Huella dactilar [2]

1.2.2 RECONOCIMIENTO DEL IRIS Y RETINA

La utilización del ojo humano en la identificación de personas ha dado lugar a dos técnicas biométricas completamente diferentes: una basada en las características del iris ocular y otra que utiliza las características distintivas de la retina, cada uno de estos sistemas tienen diferentes métodos de captura de imagen, de extracción de características y métodos de comparación.

La principal desventaja de los métodos basados en el análisis de patrones oculares radica en la escasa aceptación por parte de los usuarios ya que un examen de este órgano puede revelar enfermedades o características médicas que a muchas personas les puede interesar mantener en secreto, otro inconveniente es la utilización de lentes de contacto, ya que modifica los valores de la medida obtenida. Además, se trata de sistemas muy costosos para la mayoría de organizaciones.

1.2.2.1 Identificación por escaneo de Retina

La forma de los vasos sanguíneos es un elemento característico de cada individuo. En los sistemas de autenticación basados en patrones retinales el usuario a identificar debe mirar a través de unos binoculares, ajustar la distancia interocular y el movimiento de la cabeza, mirar a un punto determinado y por último pulsar un botón para indicar al dispositivo que se encuentra listo para el análisis. En ese momento se escanea la retina con una radiación infrarroja de baja intensidad en forma de espiral a través de la pupila, detectando los nodos y ramas del área retinal para luego compararlos con los almacenados en una base de datos, si la

[2] http://redyseguridad.fi-p.unam.mx/proyectos/biometria

muestra obtenida coincide con la almacenada se logra una identificación exitosa y se le permite el acceso al individuo.

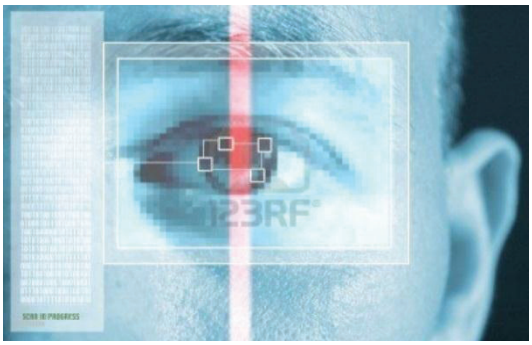


Figura 1.2: Identificación por escaneo de retina [3]

1.2.2.2 Reconocimiento del Iris Ocular

El iris humano también es considerado como una estructura única para cada individuo e inalterable durante toda la vida de la persona. Posee un sistema robusto de seguridad capaz de detectar con una alta probabilidad si el iris es natural o no en casos de intento de falsificación.

El reconocimiento de iris ocular es más moderno que el basado en patrones retinales. Su funcionamiento consiste en capturar una imagen del iris, en un entorno iluminado adecuadamente; esta imagen se somete a deformaciones pupilares (el tamaño de la pupila varía enormemente en función de factores externos, como la luz) y de ella se extraen patrones de color únicos de los surcos de la parte coloreada de nuestros ojos, que a su vez son sometidos a transformaciones matemáticas hasta obtener una cantidad de datos suficiente para los propósitos de autenticación. Esa muestra, denominada iriscode es comparada con otra tomada con anterioridad y almacenada en la base de datos del sistema, de forma que si ambas coinciden el usuario se considera autenticado con éxito.

[3] http://www.omicrono.com/2014/05/el-escaner-de-retina-de-bolsillo

Figura 1.3: Reconocimiento del iris ocular

1.2.3 RECONOCIMIENTO DE LA GEOMETRÍA DE LA MANO

Los sistemas de autenticación basados en el análisis de la geometría de la mano son sin duda los más rápidos dentro de los biométricos, en aproximadamente un segundo son capaces de autenticar a una persona. Cuando un usuario necesita ser autenticado sitúa su mano sobre un dispositivo lector con 5 clavijas que ayudan a alinear los dedos de la mano para asegurar una lectura exacta como se muestra en la figura 1.4.

Figura 1.4: Lineamiento de los dedos de la mano[4]

Una vez que la mano está correctamente situada, se hace uso de una cámara de baja resolución que captura la imagen de la palma de la mano y su sombra. En la parte izquierda de la superficie plana, se coloca un espejo formando un ángulo de 60 grados; este espejo refleja hacia la cámara el perfil lateral de la mano. Consecuentemente se extraen ciertos datos correspondientes a: anchura, longitud, área, determinadas distancias en un formato de tres dimensiones. Posteriormente estos datos son transformados en un modelo matemático que se contrasta contra una base de patrones, el sistema es capaz de permitir o negar acceso a cada usuario. Uno de los parámetros más importantes de estos sistemas de reconocimiento es que

[4] http://www.monografias.com/trabajos82/biometria

son capaces de aprender, cada vez que autentican a un usuario, actualizan su base de datos con los cambios que se puedan producir en la muestra: un pequeño crecimiento, adelgazamiento o la cicatrización de una herida; de esta forma son capaces de identificar correctamente a un usuario cuya muestra se tomó hace años, pero que ha ido accediendo al sistema con regularidad.

Una desventaja de estos autentificadores es que su tasa de falsa aceptación sea considerará inaceptable en algunas ocasiones ya que es posible, que dos personas tengan la mano lo suficientemente parecida como para que el sistema las confunda.

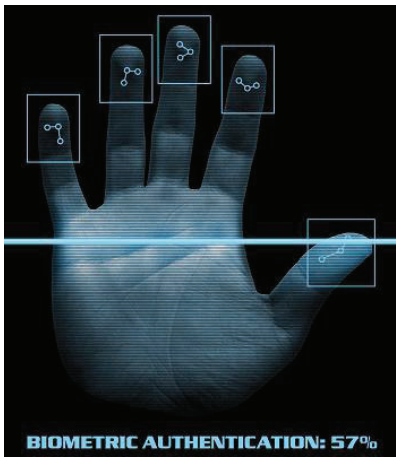


Figura 1.5: Autenticación Biométrica de la mano[5]

1.2.4 RECONOCIMIENTO DE LA FIRMA ESCRITA

Normalmente el reconocimiento de la firma escrita consiste en un análisis visual sobre una impresión en papel de los trazos de esa firma. En los sistemas biométricos se utiliza además de la forma de firmar, las características dinámicas el tiempo utilizado para rubricar, las veces que se separa el bolígrafo del papel, el ángulo de inclinación del bolígrafo, también se puede obtener información adicional como la presión instantánea ejercida a lo largo de la firma, velocidades y aceleraciones de la mano, entre otros.

El procedimiento para utilizar un sistema de autenticación de firmas consiste en solicitar un número determinado de firmas a los futuros usuarios, de las cuales el sistema extrae y almacena características propias de cada firma; esta etapa se denomina de aprendizaje, y suele presentar ciertas dificultades ya que los usuarios no suelen firmar uniformemente. Para

[5] http://dis.um.es/~lopezquesada/documentos/

combatir este inconveniente se disminuye las restricciones del sistema a la hora de aprender firmas, con lo que se decrementa su seguridad. Es un hecho que ningún sistema de identificación personal está libre de error. Sin embargo, dependiendo de la tecnología y de las condiciones de los datos tomados, este error puede ser aceptable para determinadas aplicaciones.

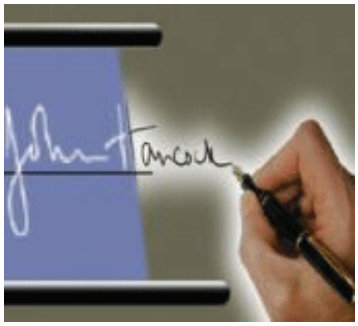


Figura 1.6: Reconocimiento de la firma escrita[6]

Una vez que el sistema conoce las firmas de sus usuarios, cuando estos desean acceder a él, se les solicita tal firma, con un número limitado de intentos generalmente más que los sistemas que autentican mediante contraseñas, ya que la firma puede variar en un individuo por múltiples factores. La firma introducida es capturada por un lápiz óptico o por una lectora sensible o por ambos como lo indica en la figura 1.6, y el acceso al sistema se permite una vez que el usuario ha introducido una firma que el verificador es capaz de distinguir y validar como auténtica.

1.2.5 RECONOCIMIENTO DE VOZ

El reconocimiento por voz, es una modalidad biométrica que utiliza la voz de un individuo con fines de reconocimiento. El proceso de reconocimiento de voz depende de las características de la estructura física del tracto vocal de una persona así como también de sus características de comportamiento. La versatilidad presente en la señal de la voz hace que el procedimiento alcance un nivel de dificultad puesto que el individuo no puede repetir de forma exacta una misma frase o palabra.

Hay dos formas de reconocimiento por voz: dependiente del texto e independiente del texto. En un sistema que utiliza discurso dependiente del texto, el individuo presenta una frase fija

[6] http://dis.um.es/~lopezquesada/documentos/

(contraseña) o una frase programada dentro del sistema ("Por favor diga los números: 33-54-63"). Un sistema independiente del texto no posee ningún conocimiento del fraseo de quien se presenta y es mucho más flexible en situaciones en las que el individuo que está dando la muestra o no está al tanto de la toma.

Las muestras de voz son ondas donde la variable del tiempo se ubica en el vector horizontal y la de volumen en el vertical. El sistema de reconocimiento de quien habla analiza el contenido de frecuencia del discurso y compara las características de calidad, duración, intensidad, dinámica y tono de la señal. La figura 1.7 indica un ejemplo de toma de muestra de la voz.

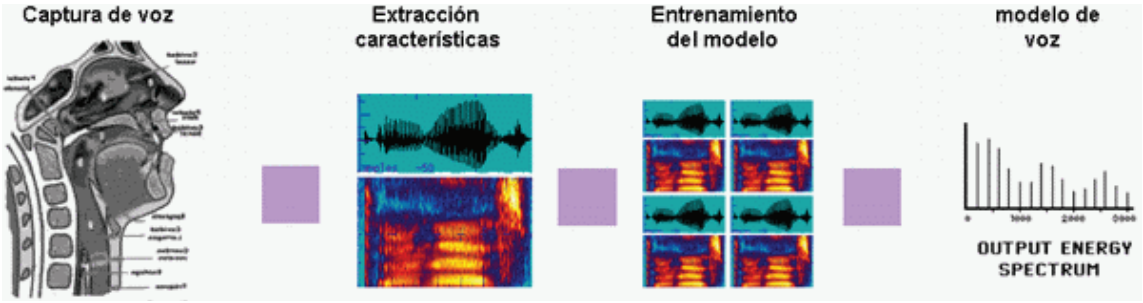


Figura 1.7: Ejemplo de toma de muestra de la voz[7]

En los sistemas dependientes del texto, durante la toma o fase de inscripción, el individuo pronuncia una palabra corta o frase (sentencia), capturada típicamente por un micrófono que puede ser tan simple como el de un teléfono. La muestra de voz se convierte desde un formato analógico a uno digital, las características de la voz del individuo se extraen, luego se crea un modelo.

Luego de la inscripción, durante la fase de reconocimiento, las mismas características de calidad, duración, volumen y tono son extraídas de la muestra tomada y comparadas con el modelo de la identidad, o hipotética identidad, y con otros modelos de otras voces. La muestra de la voz o la inscripción de sus modelos son comparados para producir un "radio de similitud", indicando la similitud por la que la muestra ingresada correspondería a la identidad buscada.

[7] http://dis.um.es/~lopezquesada/documentos/

La desventaja que presentan estos sistemas de verificación de voz, exceptuando a los que utilizan frases dadas, es que son susceptibles de ataques por spoofing a través de la utilización de una voz grabada. Para combatir estos ataques se requiere la repetición de una palabra o frase al azar. Por ejemplo, un sistema podría solicitar la repetición de una frase generada al azar, como: "33-54-63", para prevenir el ataque de una muestra de grabación.

El usuario no puede anticipar la selección azarosa de la frase y por lo tanto no puede intentar con éxito un ataque por spoofing en el sistema. La identificación mediante el reconocimiento de voz se está introduciendo muy lentamente al mercado, una de las principales razones es que este tipo de tecnología presenta altos índices de error; los sistemas tienden a tener falsos rechazos debidos principalmente a ruidos de fondo.

1.3 IDENTIFICACIÓN POR HUELLAS DACTILARES

El reconocimiento de huellas dactilares es una técnica bastante desarrollada y confiable, esta técnica se soporta en cuatro conceptos básicos fundamentales que son:

- Universalidad: El parámetro está presente en todos los seres humanos.
- Unicidad: El parámetro es único para cada persona.
- Permanencia: el parámetro no varía con el tiempo.
- Cuantificación: es posible cuantificar el parámetro.

Los sistemas de análisis biométrico en base a huellas dactilares se clasifican en: autenticación e identificación.

La categoría de autenticación consiste en obtener la imagen de la huella dactilar de un individuo, cuya identidad es conocida, para hacer una comparación con la imagen que está almacenada en la base de datos y verificar si la huella dactilar pertenece a ese individuo.

La categoría de identificación consiste en reconocer solamente la imagen de la huella dactilar y compararla con las existentes en la base de datos para hallar la identidad del individuo al que pertenece esa huella dactilar.

1.3.1 HUELLAS DACTILARES

Las huellas dactilares son rugosidades que adopta la piel de la yema de los dedos desde que el individuo se encuentra en el útero de la madre. La forma de estas rugosidades dependen de varios factores como: presión sanguínea, movimientos de la madre en el estado de gestación, posición del útero, en otras palabras el ambiente en que se encuentra el feto a lo cual se debe su diseño único.

Figura 1.8.: Huella dactilar[8]

1.3.2 IDENTIFICACIÓN DE PATRONES

A simple vista las rugosidades presentes en las yemas de los dedos son crestas (líneas) y surcos que siguen un patrón determinado único para cada dedo, por otro lado en determinados puntos las crestas se bifurcan o se cortan, estas características reciben nombre de minucias.

El conjunto de minucias presentes en la yema del dedo dan lugar a un patrón complejo único para cada individuo.

En la figura 1.9, se observa los tipos de minucias presentes en una huella dactilar.

Figura 1.9: Detalles en una huella dactilar[9]

[8] http://www.asisucede.com.mx/2014/03/24/
[9] http://www.monografias.com/trabajos57/huellas-lofoscopicas

1.4 SENSOR DE HUELLAS DACTILARES [3]

1.4.1 SENSORES ÓPTICOS

El sensor óptico es uno de los más usados en la actualidad debido a su facilidad de uso y a su bajo costo, para su funcionamiento utiliza un modelo de cámara digital CCD (Dispositivo de Carga Acoplada), en la que se encuentran miles de pixeles, cada pixel se ve estimulado con la luz incidente en él, almacenando una pequeña carga de electricidad. Los pixeles se encuentran dispuestos en forma de malla con registros de transferencia horizontales y verticales que trasladan las señales a los circuitos de procesamiento de la cámara. Esta transferencia de señales ocurre 6 veces por segundo.

Esta cámara obtiene la huella digital realizando una fotografía de la superficie del dedo.

La figura 1.10 muestra un modelo de sensor biométrico óptico.

Figura 1.10: Sensor biométrico óptico[10]

1.4.2 SENSORES CAPACITIVOS

El sensor óptico genera una imagen de las crestas y valles del dedo, utiliza un circuito integrado de silicio en cuya superficie se dispone un arreglo de pequeñas celdas, cada celda contiene dos placas conductoras cubiertas con una capa aislante. Las celdas son más pequeñas que el ancho de una cresta del dedo. El sensor es conectado a un circuito eléctrico construido sobre la base de un amplificador operacional inversor que altera un flujo de corriente. La

[10] http://es.aliexpress.com/w/wholesale-biometric-authentication-devices.html

alteración se basa en el voltaje referente de dos fuentes, generando un terminal inversor y un terminal no-inversor. El terminal no-inversor es conectado a tierra, y el terminal inversor es conectado a una fuente de voltaje de referencia y un bucle de retroalimentación que incluye las dos placas conductoras, que funcionan como un capacitor. La superficie del dedo actúa como una tercera placa capacitora, separada por las capas aislantes en la estructura de la celda y, en el caso de los valles de la huella, una bolsa de aire. La figura 1.11, muestra el funcionamiento de un sensor capacitivo.

Al producir un movimiento del dedo más cerca o más lejos de las placas conductoras, se varía la distancia entre las placas capacitoras y se cambia la capacitancia total del capacitor. Dando lugar a que el capacitor en una celda bajo una cresta tenga una capacitancia más grande que el capacitor en una celda bajo un valle. Ya que la distancia al dedo altera la capacitancia, la cresta de un dedo resultará en una salida de voltaje diferente a la del valle de un dedo.

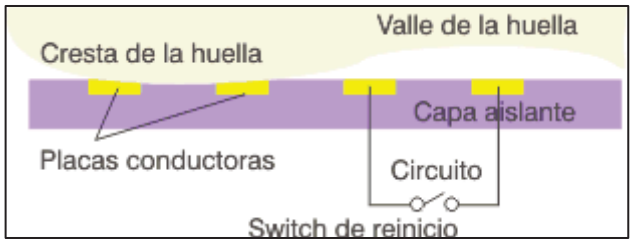


Figura 1.11: Funcionamiento de un sensor biométrico capacitivo[11]

De esta forma el procesador del lector lee esta salida de voltaje y determina si pertenece a una cresta o un valle y a partir de estos resultados puede construir una imagen de la huella, equivalente a la imagen capturada por un sensor óptico.

Este tipo de sensor es más eficiente con dedos limpios y saludables y es menos óptimo cuando se utiliza sobre pieles muy secas, callosas o con cicatrices. La humedad, la grasa y el polvo también hacen difícil el proceso de reconocimiento y reducen considerablemente la resolución de la imagen. Generalmente son más costosos que los sensores ópticos, pero son más fáciles de integrar.

[11] http://tecelectronica.com.mx/promos/bit/bit0903-bio.htm

1.4.3 SENSORES TÉRMICOS

El sensor térmico mide la temperatura diferencial entre valles (mayor calor) y crestas capilares (menor calor) de la huella digital.

Es uno de los sensores menos utilizados y actualmente en el mercado solo existe un circuito integrado para realizar estas funciones (FingerChip). Este microchip proporciona una salida dinámica en la cual el dedo se desplaza a lo largo del sensor generando una secuencia completa de imágenes, las mismas que son reconstruidas con la ayuda de un procesador para obtener una imagen completa y de alta calidad de la huella digital. La figura 1.12, muestra un modelo de sensor térmico.

El sensor dispone de una capa protectora resistente a temperaturas muy altas, humedad, suciedad, o contaminación de aceite y agua. El calentamiento del sensor es importante para evitar la posibilidad de un equilibrio térmico entre el sensor y la superficie dactilar, pero a la vez aumenta el consumo de energía considerablemente.

Figura 1.12: Sensor biométrico térmico

1.4.4 SENSORES DE CAMPO ELÉCTRICO (E-FIELD)

El sensor de campo eléctrico funciona con una antena que mide el campo eléctrico presente entre dos capas conductoras. La tecnología utilizada en este tipo de sensor origina un campo entre el dedo y el semiconductor adyacente que simula con mucha exactitud la forma de los surcos y crestas de la superficie dactilar, incluso la imagen generada es mucho más nítida que la reproducida por sensores ópticos o capacitivos.

Este sensor posee gran fortaleza para trabajar con pieles dañadas, húmedas o secas, su desventaja es que produce un índice de error alto (EER) debido a la baja resolución de la pequeña imagen.

En la figura 1.13, se muestra un modelo de sensor de campo eléctrico.

Figura 1.13: Sensor de campo eléctrico[12]

1.4.5 SENSORES SIN CONTACTO

Un sensor sin contacto funciona de forma similar al sensor óptico. Normalmente con un cristal de precisión óptica a una distancia de dos o tres pulgadas de la superficie dactilar mientras se escanea el dedo. Para poder obtener la huella dactilar se introduce la yema del dedo en un área con un hueco. Una desventaja presente en este sistema es que a través de este hueco pueden llegar polvo y suciedad hasta el cristal óptico lo cual puede producir distorsión en la lectura de la imagen. Otro punto a considerar es que las huellas escaneadas son esféricas lo que origina un algoritmo mucho más complejo.

1.5 MICROCONTROLADOR (4)

El microcontrolador consiste en un pequeño y sencillo pero completo computador contenido en un circuito integrado. Es utilizado para controlar un proceso o una terea determinada y su configuración es variable de acuerdo con la aplicación designada. A pesar de su tamaño disminuye el número de componentes en un sistema y en consecuencia el costo.

Un microcontrolador se compone de tres unidades funcionales: unidad central de proceso, memoria (de datos y de programa) y unidades de entrada y salida. La unidad central de proceso está formada por la unidad de control la cual interpreta las instrucciones y el camino de datos que las ejecuta. En la memoria de programa reside un programa destinado a gobernar una aplicación determinada. En la memoria de datos se almacena los datos del programa. Las unidades de entrada y salida soportan la conexión de sensores y actuadores del dispositivo a controlar.

[12] http://www.trazablog.com/wp-content/uploads

1.5.1 ARQUITECTURAS DE UN MICROCONTROLADOR

Existen dos tipos de arquitecturas en un computador y básicamente son las mismas en un microcontrolador estas son: Von Neumann y Harvard. La diferencia entre ambas es la forma de conexión de la memoria a la unidad central de proceso y en los buses que cada una necesita.

- Arquitectura Von Neumann

  En este tipo de arquitectura la unidad central de proceso está conectada a una memoria única donde se guardan las instrucciones de programa y de datos, teniendo un único bus por el cual se comunican. Esto hace que el microprocesador sea lento en su respuesta, ya que no es posible buscar en la memoria una nueva instrucción mientras no finalicen las transferencias de datos de la instrucción anterior.

- Arquitectura Harvard

  Esta arquitectura tiene la unidad central de proceso conectada a dos memorias independientes (una contiene solamente instrucciones de programa y la otra contiene solamente datos) por medio de dos buses totalmente independientes. Esto permite a la unidad central de proceso acceder a los datos para controlar la ejecución de una instrucción y al mismo tiempo leer la siguiente instrucción a ejecutar.

1.5.2 PARTES DE UN MICROCONTROLADOR

1.5.2.1 Procesador

Elemento encargado de direccionar, codificar y ejecutar las instrucciones desde la memoria, también es responsable de direccionar el almacenamiento de los resultados.

1.5.2.2 Registro

Es un pequeño espacio de memoria necesario en un microprocesador ya que aquí se encuentran los datos para varias operaciones que realizarán los circuitos del procesador. Estos registros guardan los resultados de la ejecución de instrucciones. Mientras los registros de datos del procesador cuenten con mayor número de bits, mayor es su velocidad de procesamiento.

1.5.2.3 Unidad de Control

La unidad de control es la encargada de coordinar y ejecutar la mayoría de las operaciones en el procesador, es decir interpretar las instrucciones generadas por un programa y después iniciar las acciones correspondientes para ejecutar dichas instrucciones es decir, realiza el control de los registros, la ALU, buses de direcciones y datos y partes pertinentes del procesador. De este modo la unidad de control determina el rendimiento del procesador en aspectos como: velocidad de ejecución, tipos de buses del sistema, tiempo del ciclo de máquina, etc.

1.5.2.4 Unidad Aritmética Lógica

Es la unidad encargada de realizar todas las operaciones lógicas (comparación entre números) y operaciones matemáticas (suma, resta, multiplicación, división, referentes al algebra de Boole).

1.5.2.5 Memoria

Es una unidad de almacenamiento integrada en el microcontrolador.

1.5.2.6 Tipos de memoria

1.5.2.6.1 Memoria de datos

RAM (Random Access Memory). Es una memoria de tipo volátil, es decir, de almacenamiento temporal, la información permanecerá en la memoria hasta que el microcontrolador disponga de energía eléctrica.

1.5.2.6.2 Memoria de programa

- ROM (Read Only Memory): Este tipo de memorias se fabrica con los datos almacenados de forma permanente, es decir, esta memoria permite únicamente la lectura de la información disponga o no de energía eléctrica el microcontrolador.
- OTP (One Time Progammable): Este tipo de memoria además de permitir la lectura de la información, permite programar a la memoria una sola vez, es decir, grabar información en ella lo cual se realiza mediante un grabador desde un computador.
- EPROM (Erasable Programmable Read Only Memory): Esta memoria funciona igual que la OTP pero con la excepción de que en la memoria EPROM se puede borrar y grabar información varias veces.

- EEPROM (Electrical EPROM): Es un tipo de memoria de lectura, también es posible programar y borrar la memoria eléctricamente, este proceso se realiza con el microcontrolador instalado en el circuito o con un grabador desde un computador.
- Memoria flash: derivada de la memoria EEPROM, es un tipo de memoria no volátil puede borrarse y grabarse eléctricamente cuantas veces sea necesario pero además la memoria flash cuenta con un bajo consumo de energía eléctrica, mayor velocidad, y su tamaño es reducido.

1.5.2.7 Buses

Al transmitir información entre los diferentes componentes del procesador, se lo hace a través de los buses que no son más que un medio de comunicación.

Existen tres tipos:

- Bus de dirección: Establece la dirección de memoria del dato en tránsito.
- Bus de datos: Transfiere datos entre los componentes del procesador.
- Bus de control: Gestiona los procesos de lectura y escritura y controla la correcta operación de los componentes del sistema.

1.5.2.8 Puertos de entrada y salida

Son puertos de comunicación con los que cuenta el microntrolador, se encuentran en determinados pines del mismo, su principal función es soportar las señales de control, entrada y salida.

1.5.2.9 Reloj

Es un circuito oscilador generalmente incorporado en el microcontrolador, este circuito genera pulsos cuadrados de alta frecuencia los cuales sincronizan las operaciones del sistema. Para poder estabilizar la frecuencia de trabajo el circuito necesita de un cristal de cuarzo y una red R-C, debe tomarse en cuenta que aumentar la frecuencia de oscilación del circuito conlleva a disminuir el tiempo con que se ejecutan las instrucciones pero también aumenta considerablemente la temperatura del microcontrolador y el consumo de energía eléctrica.

1.6 TABLET [5]

Una Tableta o Tablet, es una computadora portátil orientada a la multimedia, lectura de contenidos y a la navegación web que a usos profesionales, es de mayor tamaño que un smartphone, integrada en una pantalla táctil (sencilla o multitáctil) con la que se interactúa primariamente con los dedos, sin necesidad de teclado físico ni otros componentes como el mouse. Estos últimos se ven reemplazados por un teclado virtual y, en determinados modelos, se utilizan los denominados lápices capacitivos.

Dependiendo del sistema operativo que implementen y su configuración, al conectarse por USB a un ordenador, se pueden presentar como dispositivos de almacenamiento, mostrando solo la posible tarjeta de memoria conectada, la memoria flash interna, e incluso la flash ROM.

1.6.1 VENTAJAS Y DESVENTAJAS DEL USO DE TABLETS.

Las siguientes ventajas y desventajas se obtienen en comparación entre las tabletas y los computadores portátiles:

1.6.1.1 Ventajas

- Su facilidad de uso en entornos donde resulta complicado un teclado y un ratón, como el manejo con una sola mano.
- Su peso ligero. Los modelos de menor potencia pueden funcionar de manera similar a los dispositivos de lectura.
- El entorno táctil hace que en ciertos contextos como en la manipulación de imágenes, música o juegos, el trabajo sea más fácil que con el uso de un teclado y un ratón.
- Para algunos usuarios resulta más interactivo y agradable usar un lápiz, una pluma o el dedo para apuntar y pulsar sobre la pantalla, en lugar de utilizar un ratón o un touchpad.
- La duración de la batería es mucho mayor que la de una computadora portátil.

1.6.1.2 Desventajas

- Velocidad de interacción: la escritura a mano sobre la pantalla, o escribir en un teclado virtual, puede ser significativamente más lento que la velocidad de escritura en un teclado convencional, que puede llegar hasta las 50 a 150 palabras por minuto.
- Riesgos en la pantalla: las pantallas de las tabletas se manipulan más que las de los portátiles convencionales, sin embargo, muchas están fabricadas de manera similar. Además, puesto que las pantallas también sirven como dispositivos de interacción, corren un mayor riesgo de daños debido a los golpes y al mal uso.
- Comodidad (ergonomía): una tableta no ofrece espacio para el descanso de la muñeca. Además, el usuario tendrá que mover su brazo constantemente mientras escribe.

1.6.1.3 Sistemas Operativos

Las tabletas, al igual que los computadores tradicionales, pueden funcionar con diferentes sistemas operativos. Estos se dividen en dos clases:

- Sistemas operativos basados en el escritorio de un computador tradicional y
- Sistemas operativos pos-PC (similares a los SO de los teléfonos móviles inteligentes).

Para la primera clase, los sistemas operativos más populares son el Windows de Microsoft y una variedad de sistemas de Linux. HP está desarrollando tabletas orientadas a las necesidades empresariales basadas en Windows y tabletas orientadas al consumidor personal basadas en webOS.14 15.

Para la segunda clase, los sistemas operativos más populares incluyen el iOS de Apple y el Android de Google. Muchos fabricantes también están probando productos con Windows 8, con el Chrome OS de Google y otros.

1.7 COMUNICACIÓN SERIAL [6]

1.7.1 INTRODUCCIÓN

La comunicación serial es utilizada para la comunicación entre dispositivos que se incluye en cualquier computadora o en microcontroladores de alto rendimiento. Consiste en el envío de un bit de información de manera secuencial, esto es, un bit a la vez y a un ritmo acordado entre el emisor y el receptor.

Existen en la actualidad diferentes ejemplos de puertos que comunican información de manera serial. El conocido como "puerto serial" ha sido gradualmente reemplazado por el puerto USB (Universal Serial Bus) que permite mayor versatilidad en la conexión de múltiples dispositivos. La mayoría de los microcontroladores, poseen un puerto de comunicación serial. Para comunicarse con los computadores personales que poseen únicamente puerto USB se requiere de un dispositivo "traductor" como el integrado FT232RL, el cual es un convertidor USB-Serial. A través de este integrado el microcontrolador puede recibir y enviar datos a un computador de manera serial.

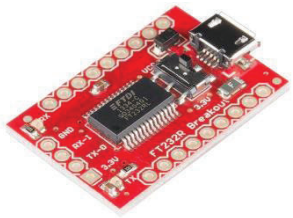


Figura 1.14: Convertidor USB-Serial[13]

Comúnmente la comunicación serial se utiliza para transmitir datos en formato ASCII. Para establecer la comunicación se utilizan 3 líneas de transmisión:

1) Tierra o referencia, 2) Transmisión, 3) Recepción. Debido a que la transmisión es asincrónica, es posible enviar datos por una línea mientras se reciben datos por otra. Existen otras líneas disponibles para realizar handshaking, o intercambio de pulsos de sincronización, pero no son requeridas. Los parámetros más importantes de la comunicación serial son la velocidad de transmisión, los bits de datos, los bits de parada, y la paridad teniendo en cuenta que para que dos puertos se puedan comunicar, es necesario que las características sean iguales.

1.7.1.1 Velocidad de transmisión (baud rate)

Indica el número de bits por segundo que se transfieren, y se mide en baudios (bauds). Por ejemplo, 300 baudios representan 300 bits por segundo. Cuando se hace referencia a los ciclos de reloj se hace referencia a la velocidad de transmisión. Por ejemplo, si el protocolo hace una llamada a 4800 ciclos de reloj, entonces el reloj está corriendo a 4800 Hz. Las altas velocidades se utilizan cuando los dispositivos se encuentran uno junto al otro.

[13] http://www.eneka.com.uy/robotica/tarjetas-accesorias-de-desarrollo.html

1.7.1.2 Bits de datos

Se refiere a la cantidad de bits en la transmisión. Cuando la computadora envía un paquete de información, el tamaño de ese paquete no necesariamente será de 8 bits. Las cantidades más comunes de bits por paquete son 5, 7 y 8 bits. El número de bits que se envía depende del tipo de información que se transfiere. Por ejemplo, el ASCII estándar tiene un rango de 0 a 127, es decir, utiliza 7 bits; para ASCII extendido es de 0 a 255, lo que utiliza 8 bits. Si el tipo de datos que se está transfiriendo es texto simple (ASCII estándar), entonces es suficiente con utilizar 7 bits por paquete para la comunicación. Un paquete se refiere a una transferencia de byte, incluyendo los bits de inicio/parada, bits de datos, y paridad.

1.7.1.3 Bits de parada

Usado para indicar el fin de la comunicación de un solo paquete. Los valores típicos son 1, 1.5 o 2 bits. Debido a la manera como se transfiere la información a través de las líneas de comunicación y que cada dispositivo tiene su propio reloj, es posible que los dos dispositivos no estén sincronizados. Por lo tanto, los bits de parada no sólo indican el fin de la transmisión sino además dan un margen de tolerancia para esa diferencia de los relojes. Mientras más bits de parada se usen, mayor será la tolerancia a la sincronía de los relojes, sin embargo la transmisión será más lenta.

1.7.1.4 Paridad

Es una manera sencilla de verificar si hay errores en la transmisión serial. Existen dos tipos de paridad: par, impar. La opción de no usar paridad alguna también está disponible. Para paridad par e impar, el puerto serial fijará el bit de paridad (el último bit después de los bits de datos) a un valor para asegurarse que la transmisión tenga un número par o impar de bits en estado alto lógico. Por ejemplo, si la información a transmitir es 011 y la paridad es par, el bit de paridad sería 0 para mantener el número de bits en estado alto lógico como par. Si la paridad seleccionada fuera impar, entonces el bit de paridad sería 1, para tener 3 bits en estado alto lógico.

CAPÍTULO 2

IMPLEMENTACIÓN DE HARDWARE Y SOFTWARE DEL SISTEMA.

2.1 DIAGRAMA DE BLOQUES DEL SISTEMA

En el siguiente diagrama de bloques se muestra los dispositivos que intervienen en el sistema.

La comunicación entre el microprocesador, la Tablet y el sensor biométrico se realiza a través del UART (Transmisor-Receptor Asíncrono Universal), para que los datos en formato serie puedan ser transmitidos a través de los puertos.

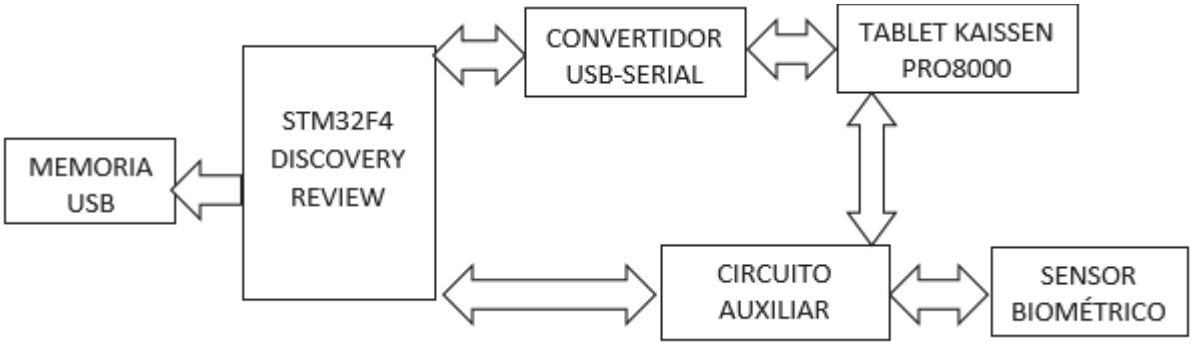


Figura 2.1: Diagrama de bloques del sistema

2.2 DISPOSITIVOS UTILIZADOS EN EL SISTEMA

2.2.1 STM32F4 DISCOVERY REVIEW [7]

Esta placa es un sistema embebido el cual permite descubrir las características del microcontrolador STM32F407 VGT6 y desarrollar aplicaciones.

Es considerado el elemento principal del sistema, ya que se encarga de enviar y recibir datos entre los dispositivos interconectados (Tablet y Sensor Biométrico) a través de la comunicación serial. Posee interfaces para la conexión de la Memoria USB y demás características que se describen a continuación.

Figura 2.2: STM32F4 Discovery Review[14]

2.2.1.1 Características del STM32F4 Discovery Review

2.2.1.1.1 ST-Link

Un incrustado ST-LINK está incluida en la placa de evaluación STM32F4-Discovery que se puede utilizar para programar el microcontrolador STM32F407VG a bordo.

2.2.1.1.2 Periféricos y Soportes

- **Dispositivos I / O Interface:** la placa cuenta con cuatro LEDs y un interruptor pulsador programables por el usuario. Un segundo botón pulsador está conectado a la patilla de reposición (NRST) del microcontrolador.
- **Audio: e**l STM32F407VG es también conectado a un DAC de audio y micrófono MEMS. Una clavija estéreo de audio está a bordo para conectar auriculares o altavoces a la salida de audio de la DAC.
- **USB:** la tarjeta dispone de un puerto USB OTG.
- **Sensor de movimiento:** un acelerómetro de tres ejes lineales.
- **Headers:** todos los pines I / O de la STM32F407VG están disponibles en conectores macho. 2 x 25 (50 pin) con su cabecera ubicada a cada lado de la placa a lo largo de su longitud. Los nombres de los pines son serigrafiados al lado de los conectores macho.

[14] Especificaciones Técnicas STM32F4 DISCOVERY

- **Osciladores:** aunque la STM32F407VG tiene dos osciladores internos RC, utiliza un cristal de 8MHz para el oscilador principal.
- **Puentes de soldadura:** la placa tiene un número de conectores de puente de soldadura que pueden ser soldados o cerrados y desoldados para abrirlos con el fin de hacer o deshacer las conexiones.
- **Consumo de corriente:** si se necesita medir el consumo de corriente del microcontrolador, se puede quitar un puente de la placa para conectar un amperímetro.

2.2.1.2 Características del microcontrolador STM32F407VG

Soldado en la placa se puede observar un microcontrolador STM32F407V. Este microcontrolador contiene un núcleo ARM Cortex-M4 con FPU (unidad de punto flotante).

El microcontrolador STM32F407VG es el encargado de procesar todas las acciones y ejecutar las instrucciones programadas.

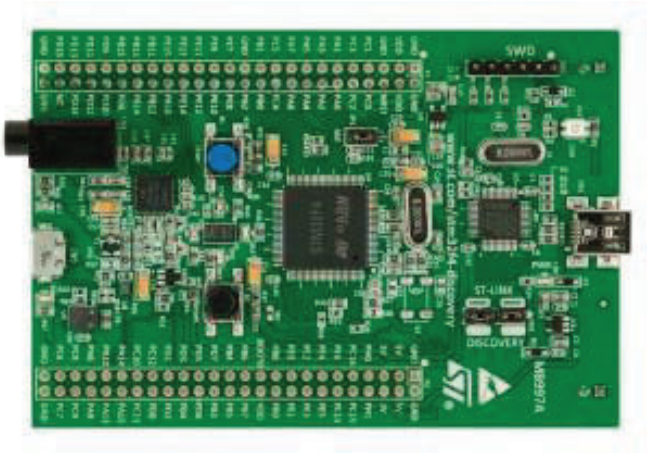


Figura 2.3: STM32F4 Discovery Review

Algunas de las características del microcontrolador STM32F407VG son:

- 1 MByte de memoria flash
- 192KBytes de RAM
- Oscilador RC
- Alimentación por una sola fuente de 1.8V a 3.6V
- Puede operar a una velocidad de 168MHz

• Pines I / O multiplexados con muchos periféricos internos

• Puerto USB OTG

• Ethernet

• Controlador de memoria estática de soporte Compact Flash, SRAM, PSRAM, NOR y NAND

• Interfaz paralela para LCD

2.2.1.3 Requisitos de hardware

El único elemento de hardware necesario para empezar a programar el microcontrolador es un cable de conexión USB A a mini-B y un puerto USB disponible en su PC. Si desea utilizar el puerto USB OTG necesitará un cable USB de tipo A a micro-B.

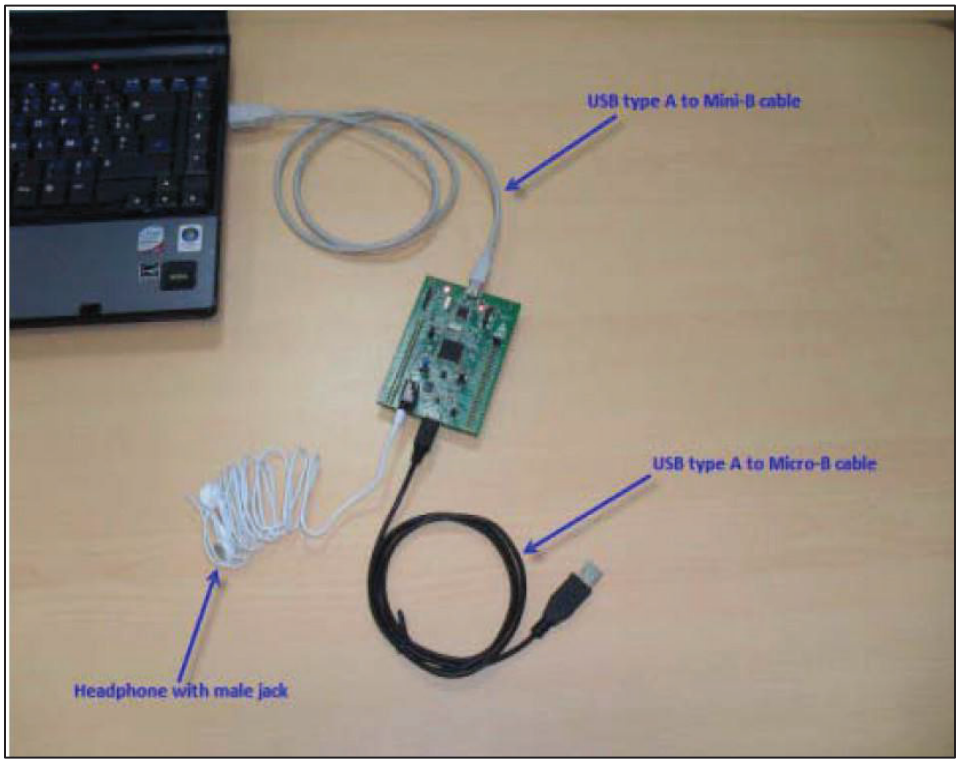


Figura 2.4: Elementos de conexión para el STM32F4 Discovery Review

2.2.1.4 Requisitos de software

Será necesaria una plataforma de Windows para ejecutar las herramientas de desarrollo para microcontroladores STM32.

Alternativamente, existen aplicaciones de código abierto como CoIDE-1.7.7 para Windows, que puede ser usado para programar los microcontroladores STM32.

Una parte de la placa STM32F4DISCOVERY es un depurador ST-LINK/V2 que es soportada por diversos entornos de desarrollo.

2.2.1.5 Herramientas gratuitas

- GNU toolchain ARM
- CoIDE-1.7.7
- YAGARTO (ARM GNU toolchain para Windows.

2.2.1.6 Herramientas comerciales

- Atollic TrueSTUDIO profesional para STM32
- IAR Embedded Workbench para ARM
- Keil MDK-ARM

2.2.1.7 Ventajas

- Bajo costo
- Incluye un ST-LINK embebido para programar y depurar
- Alimentado por USB para una evaluación rápida y fácil
- Excelente documentación

2.2.2 SENSOR BIOMÉTRICO ZFM-20 [8]

Los Sensores Biométricos de la serie ZFM-20 son módulos de identificación de huellas digitales fabricados por Hangzhou Zhian Technologies Co., Ltd., que tiene Synochip DSP como el procesador principal.

El módulo realiza una serie de funciones como la toma de la huella digital, enrolamiento, procesamiento de imágenes, búsqueda y almacenamiento de plantillas.

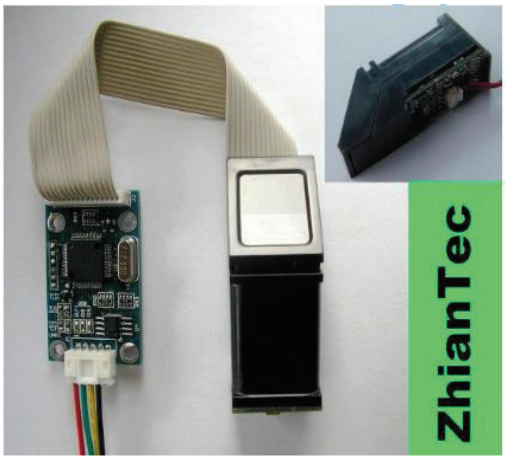


Figura 2.5: Sensor Biométrico ZFM-20[15]

2.2.2.1 Principio de funcionamiento

El procesamiento de huellas dactilares incluye dos partes: el registro de huella e identificación (el proceso puede ser de 1: 1 o 1: N).

En el enrolamiento, el usuario coloca el dedo en el sensor óptico y el sistema generará una plantilla del dedo y lo comparará con las plantillas almacenadas en la biblioteca.

En el proceso 1: 1, el sistema comparará la imagen del dedo capturada con la plantilla específica designada en el módulo; el proceso de 1: N, el sistema buscará la imagen del dedo en toda la biblioteca. En ambas circunstancias, el sistema devolverá el resultado de la verificación: exitoso o fallido.

2.2.2.2 Parámetros Principales

Alimentación	DC 3.6V-6.0V	Interfaz	UART (TTL lógico nivel) / USB 1.1
Corriente de trabajo	Típico: 100mA Pico: 150mA	Modo de trabajo	1:1 and 1:N
Velocidad de transmisión	(9600 * N) bps, N = 1 ~ 12(default N=6)	Tamaño de archivo de caracteres	256 bytes
Tiempo adquiriendo Imagen	<1s	Tamaño de la plantilla	512 bytes

[15] Especificaciones Técnicas ZMF-20 SERIES FINGER

Capacidad de almacenamiento	120/ 375/ 880	Nivel de seguridad	5 (1, 2, 3, 4, 5(highest))
FAR	<0.001%	FRR	<0.1%
Tiempo medio de búsqueda	< 1s (1:880)	Dimensión Ventana	14mm*18mm
Ambiente de trabajo	Temp: -10℃- +40℃ RH: 40%-85%	Entorno de almacenamiento	Temp: -40℃- +85℃ RH: <85%
Contorno de Dimensión	Tipo Split	Módulo: 42*25*8.5mm (Dimensión instalado: 31.5*19mm) Sensor:56*20*21.5mm	
	Tipo integral	56*20*21.5mm	

2.2.2.3 Interfaz de hardware

Para la conexión con el ordenador superior (PC) se hace uso de una comunicación serial mediante una interfaz UART o USB.

- **Comunicación en serie**

Definición de pines del biométrico (J1)

Número de pin	Nombre	Tipo	Descripción
1	Vin	in	Entrada de Alimentación（cable color: rojo）
2	TD	out	Salida de Datos nivel lógico TTL (cable color: verde）
3	RD	in	Entrada de Datos nivel lógico TTL (cable color: blanco)
4	GND	—	Señal Tierra /Conectada a Tierra (cable color negro）
5	NC	—	No conectada.

- **Conexión del Hardware**

El módulo puede comunicarse con el MCU de 3.3V o 5V vía interfaz serial de la siguiente manera:

TD (pin 2 del J1) se conecta con RXD (pin de recepción del MCU).

RD (pin 3 de J1) se conecta con TXD (pin de transferencia del MCU).

2.2.3 TABLET KAISSEN PRO 8000

La Tablet KAISSEN PRO 8000 es utilizada como interfaz gráfica para el usuario, mediante la cual el docente podrá ingresar el tema a dictarse en clase.

Se ha elegido este modelo específico debido a que nos proporciona los parámetros necesarios para el correcto funcionamiento del Sistema de Registro; así como también provee una mayor velocidad de procesamiento en comparación con otras marcas disponibles en el mercado.

2.2.3.1 Principio de funcionamiento

Cuenta con un sistema operativo intuitivo familiar Android OS 4.0 de Google con un potente procesador Dual Core para mayor rapidez.

Figura 2.6: Tablet KAISSEN PRO 8000[16]

2.2.3.2 Parámetros Principales

Touch panel con una pantalla de excelente resolución 1024 x 768 px.

Dimensiones: 21.4 x 16.1 x 1.27 cms.

Información de la batería

Batería recargable 120 horas o 5 días en Standby

Hasta 5 horas reproduciendo video

[16] http://www.miroenlinea.com/producto/

Interfaces/Puertos

HDMI

Tipo de conexión: Mini USB

Almacenamiento

60 bit NAND flash de almacenamiento

Memoria de Sistema de 1GB

Memoria interna de 4GB-32GB (default 4GB)

Soporta Micro SD card hasta 32GB

Soporta memoria DDR3

Red & Comunicación

GPS integrado

Wireless LAN

3G Móvil integrado

Tecnología de conectividad: UMTS, HSDPA, GPRS, EDGE

Bluetooth

En el equipo

Micrófono

Speakers

Cámara Web frontal 0.3M Px

Cámara Principal posterior 2M Px

Software

Sistema Operativo Android 4.0 Ice Cream Sandwich

2.2.4 CIRCUITO AUXILIAR [9]

Para incorporar la memoria Micro SD al Microcontrolador, se ha desarrollado un circuito auxiliar, al mismo tiempo el circuito facilita la interconexión del Microcontrolador con la Tablet, el Sensor Biométrico y la Micro SD para la transmisión y recepción de datos.

La Micro SD permite almacenar información referente al registro académico que posteriormente puede ser extraída utilizando una memoria USB, consecuentemente esta información puede ser leída en formato texto en una PC.

La figura 2.7 muestra el diseño del circuito en Proteus:

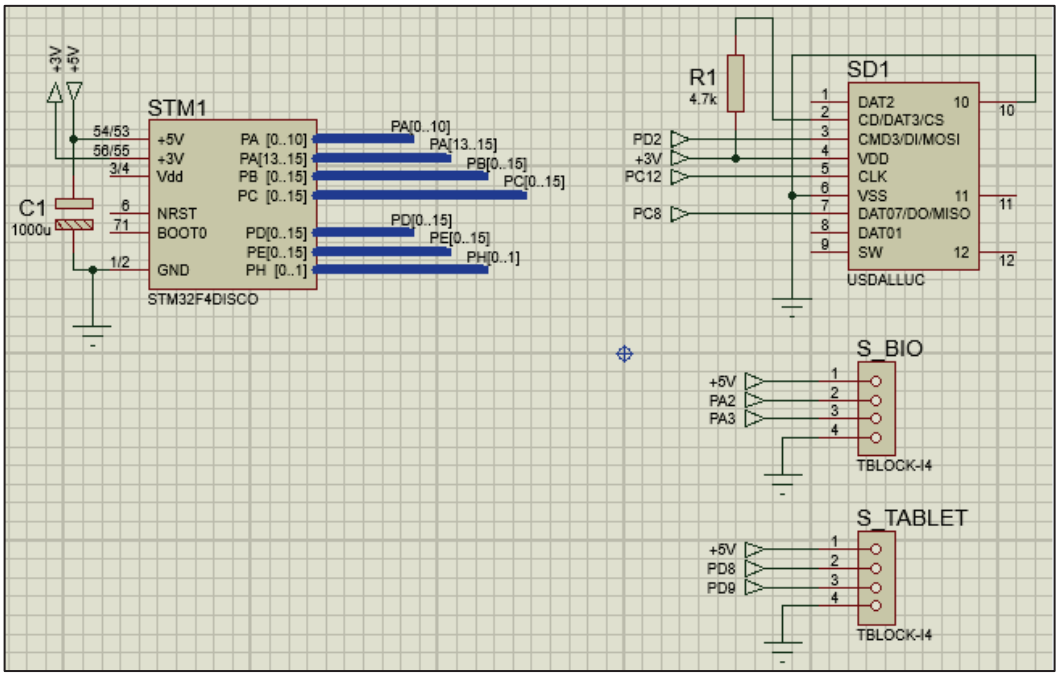


Figura 2.7: Circuito Auxiliar

El diagrama está compuesto por el microcontrolador principal STM32F40, el cual hace uso de un bus de datos para poder intercambiar información con la Tablet, el Sensor Biométrico y la Micro SD, también incluidos en el diagrama.

Cada elemento posee pines destinados para la conexión con el Microcontrolador.

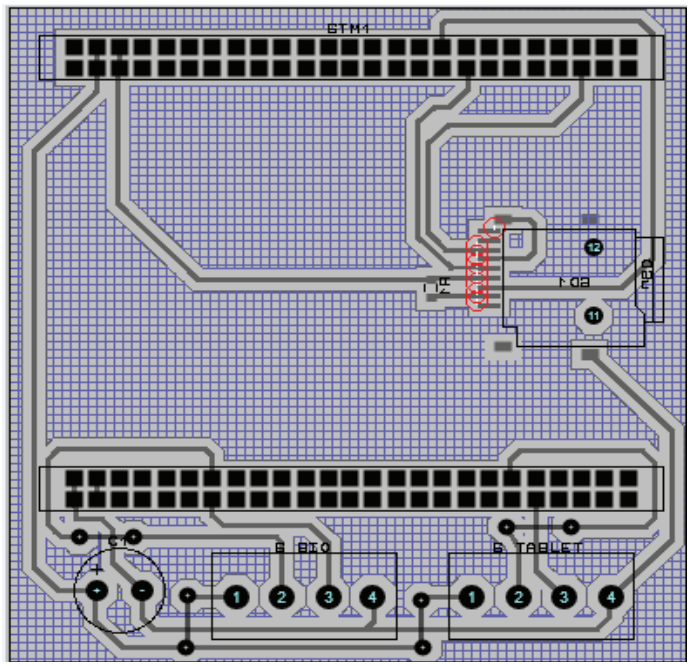


Figura 2.8: Circuito Auxiliar impreso

2.2.5 CONVERTIDOR USB-SERIE

Dado que el STM32 trabaja con un nivel de voltaje de 3,3 V, será necesario un componente de cambio de nivel para conectar el USART del STM32 a cualquier puerto serie del PC. Para esto usamos el FT232RL FTDI que es un dispositivo convertidor de USB a serie basado en un chip FTDI, básicamente se utiliza cuando se va a conectar los USARTs de un microcontrolador a un PC.

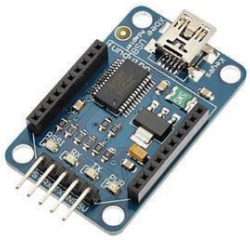


Figura 2.9: FT232RL FTDI[17]

[17] http://es.aliexpress.com

2.2.5.1 Conexión del STM32 con el Convertidor FT232RL.

El STM32 Discovery cuenta con pares de conectores de extensión que se pueden utilizar para transmisión serial USART (Transmisor Receptor Universal Asincrónico y Sincrónico.).

Mediante el uso de estos pares de conectores, el STM32 Discovery puede ser conectado al puerto serie (USB) de un PC. En la siguiente figura se muestra la conexión del STM32 con el FT232RL.

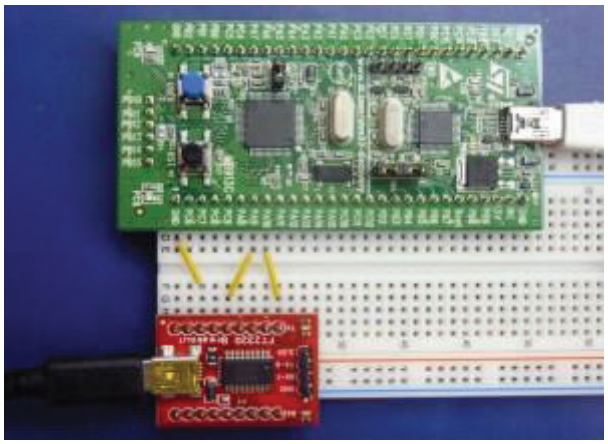


Figura 2.10: STM32 conectado a FT232RL

Esta conexión es bastante simple, sólo necesitamos los pines RX/TX de la STM32 y el FT232RL. A continuación se muestra los pines externos para comunicación serial con los que cuenta el STM32 Discovery.

USART1 : TX:[PA9,PB6] RX:[PA10,PB7]

USART2 : TX:[PA2,PD5] RX:[PA3,PD6]

USART3 : TX:[PB10,PC10,PD8] RX:[PB11,PC11,PD9]

UART4 : TX:[PA0,PC10] RX:[PA1,PC11]

UART5 : TX:[PC12] RX:[PD2]

USART6 : TX:[PC6,PG14] RX:[PC7,PG9]

Cualquiera de estos pares de pines se puede utilizar para una aplicación de comunicación serial, por ejemplo si deseamos utilizar USART1, debemos cablear de la siguiente manera:

PA9 del STM32 (USART1_TX) se conecta al pin RXD del FT232RL.

PA10 del STM32 (USART1_RX) se conecta al pin TXD del FT232RL.

GND del STM32 se conecta a GND del FT232RL.

2.2.5.1.1 Software para la verificación de la conexión

Por el lado de PC se utiliza un programa de terminal serie para enviar y recibir caracteres desde y hacia el STM32.

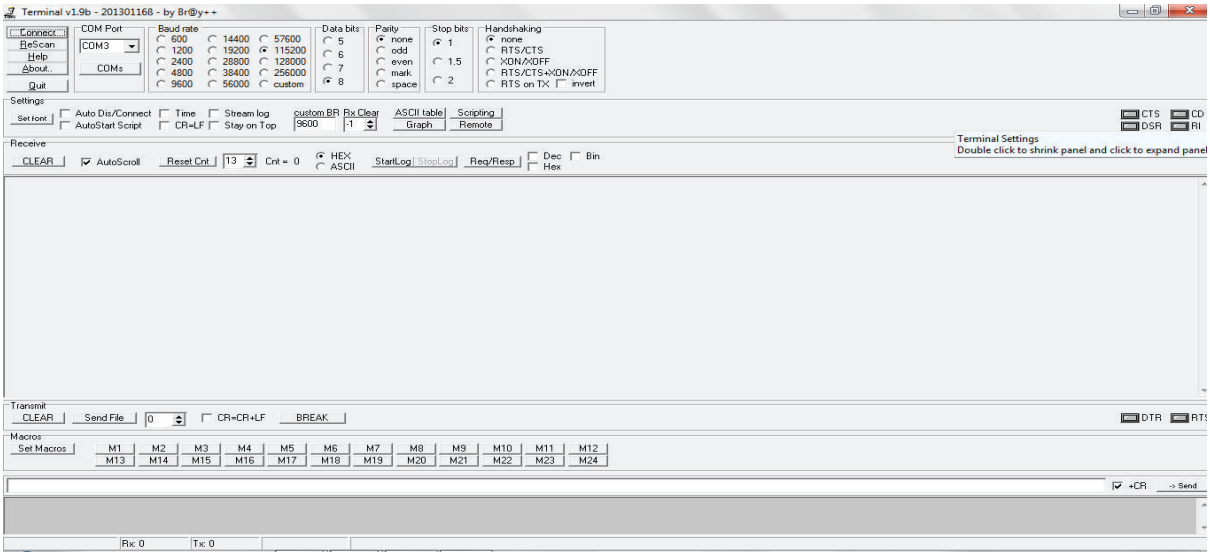


Figura 2.11: Terminal v1.9b

También se debe instalar en la PC un driver (CP210x_VCP_Windows_Driver) el cual le permite reconocer al FT232RL FTDI y trabajar con él.

2.3 DESARROLLO DEL SOFTWARE PARA EL SISTEMA ELECTRÓNICO

2.3.1 DIAGRAMA DE FLUJO DEL REGISTRO DE CUMPLIMIENTO ACÁDEMICO

El diagrama de flujo es la representación gráfica de un algoritmo. Estos diagramas utilizan símbolos con significados definidos que representan los pasos del algoritmo, y mediante flechas indican la dirección de ejecución del proceso.

Son utilizados para facilitar la comprensión de un proceso; así como también para reemplazar varias hojas de texto.

Mediante el siguiente diagrama se representa la lógica utilizada para la ejecución del sistema de Registro de Cumplimiento Académico.

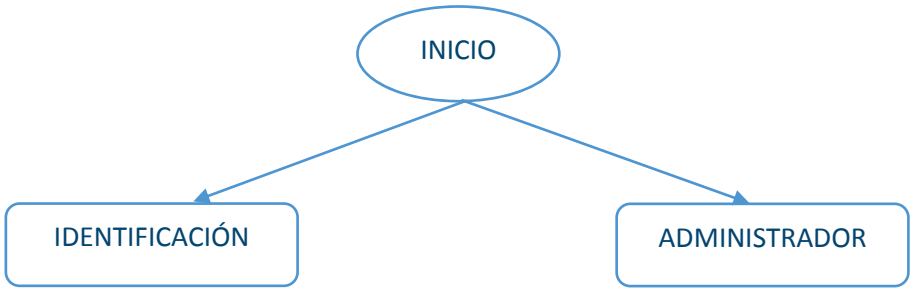


Figura 2.12: Inicio del flujograma

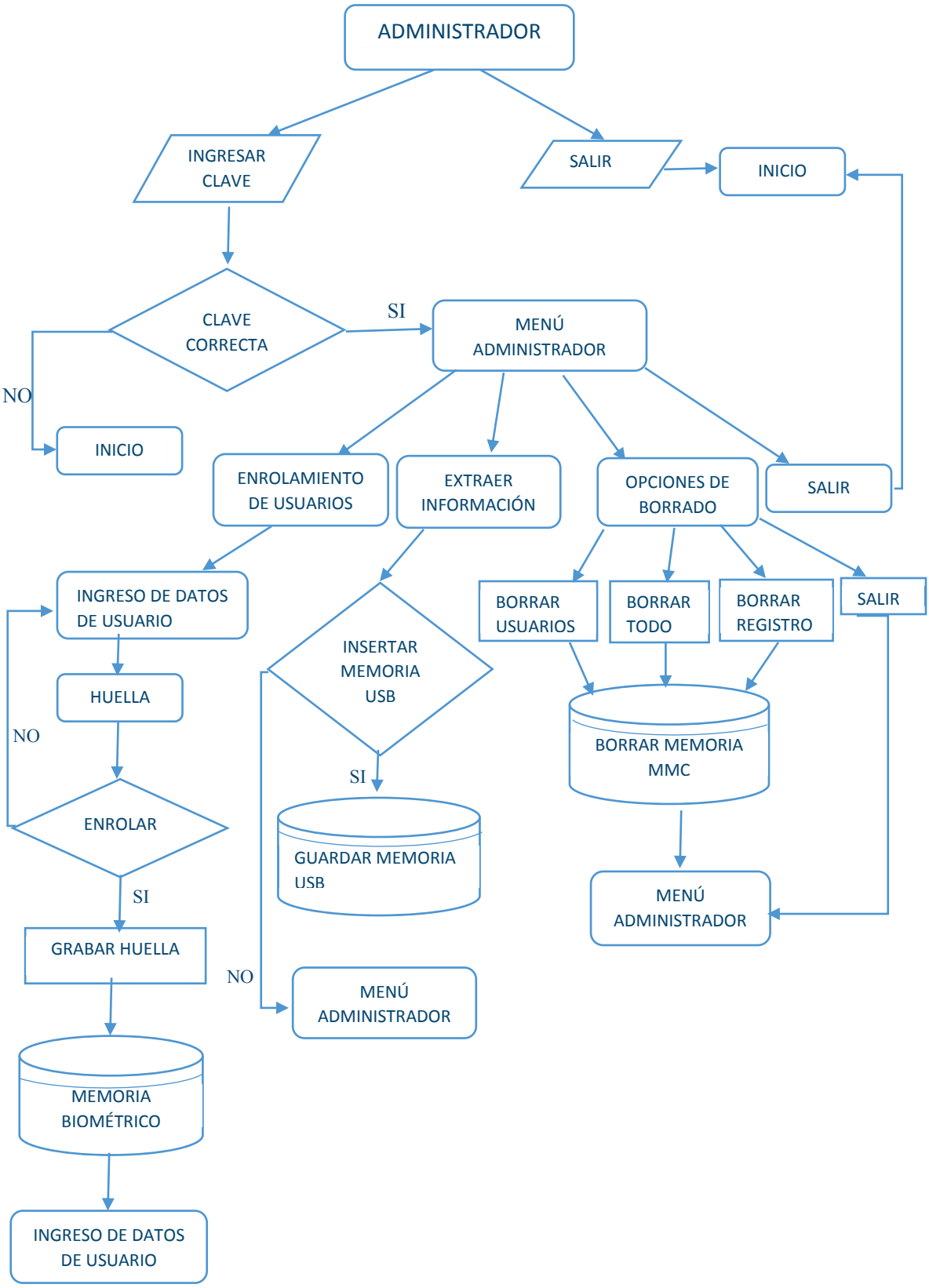


Figura 2.13: Diagrama de flujo del Administrador

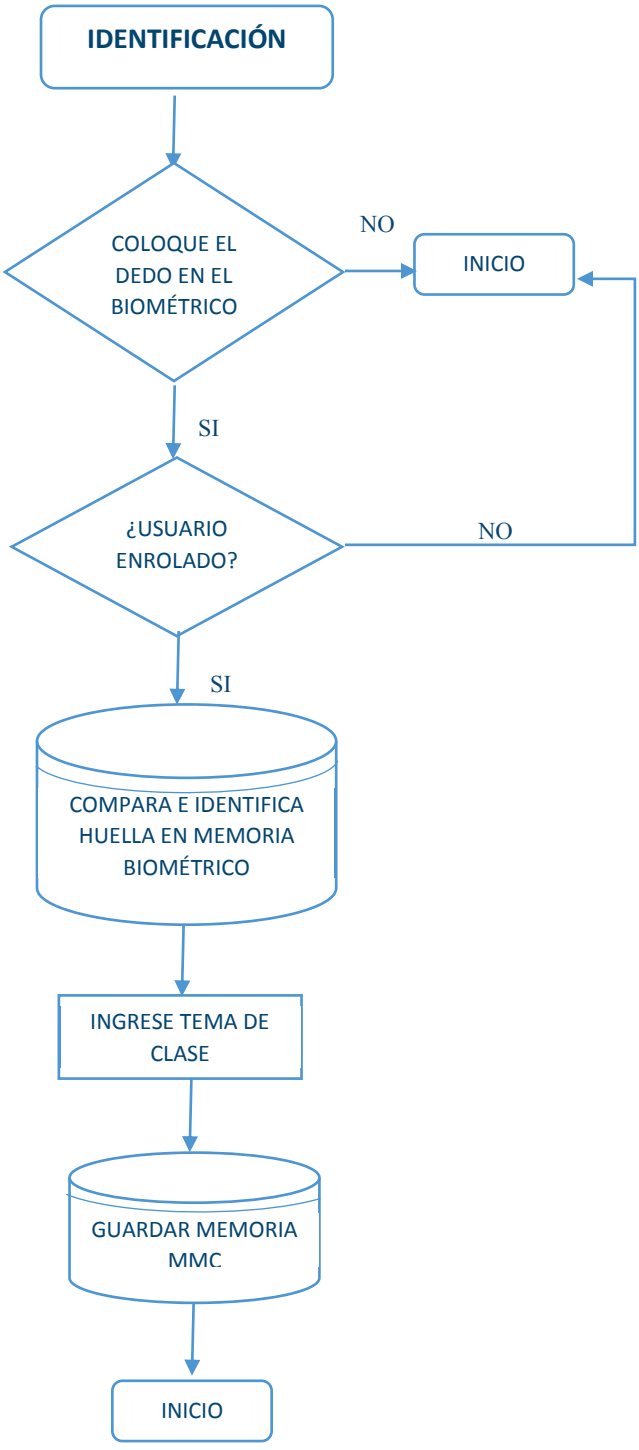


Figura 2.14: Diagrama de flujo del Usuario

2.3.1.1 Descripción del Diagrama de flujo

- El programa se inicia con dos opciones para el usuario que son: IDENTIFICACIÓN y ADMINISTRADOR.

IDENTIFICACIÓN:

- Si el usuario elige la opción de IDENTIFICACIÓN, el dispositivo pedirá que se coloque el dedo en el sensor biométrico para poder autenticar e identificar al individuo. Si no se coloca el dedo en el lector biométrico en el tiempo establecido, el sistema volverá automáticamente al inicio.

- Una vez que se haya colocado el dedo en el biométrico, este lee la huella y busca en su base de datos para comprobar si el individuo se encuentra enrolado.

- Si el individuo se encuentra previamente enrolado, luego de obtener la imagen de su huella dactilar, en el proceso de autenticación se hace una comparación con la imagen que está almacenada en la base de datos y se verifica si la huella dactilar pertenece a ese individuo. Caso contrario el sistema emitirá un mensaje de "Usuario no admitido" y volverá a la pantalla inicial.

- Posteriormente el sensor biométrico realiza el proceso de identificación que consiste en reconocer solamente la imagen de la huella dactilar y compararla con las existentes en la base de datos para hallar la identidad del individuo al que pertenece esa huella dactilar.

- Una vez que se ha autenticado e identificado al usuario, el dispositivo permitirá ingresar el tema de clase.

- Finalmente con la opción de GUARDADO se guardará toda la información ingresada por el docente, incluyendo la fecha y hora de registro.

ADMINISTRADOR

- Cuando se elija la opción de ADMINISTRADOR, el sistema pedirá que se ingrese la clave de Administrador, mediante el teclado de la Tablet.

- Si la clave es correcta, a continuación se desplegará en la pantalla un menú en el que se encuentran como opciones:
  Enrolamiento de Usuarios
  Extraer Información
  Opciones de Borrado

- El Enrolamiento de Usuarios permite ingresar los datos de los usuarios (Nombre, Hora1, Hora2, Código, Materia, Paralelo, Aula), seguido por la huella digital del docente a registrar.

- El sistema enrola al usuario y guarda la huella en la memoria del sensor biométrico.

- Con la opción de Extraer Información, el sistema muestra en la pantalla el mensaje "Insertar Memoria USB". Con la opción EXTRAER se extrae y se guarda la información en la Memoria USB.

- Se tiene también la opción OPCIONES DE BORRADO que permite:
  Borrar Usuarios: con la cual se puede borrar un usuario a la vez de la MMC.
  Borrar Todo: con esta opción se pueden borrar a todos los usuarios registrados de la MMC.
  Borrar Registro: se la utiliza cuando se desea borrar todos los registros almacenados en la MMC.
  Regresar: se la usa para volver al Menú del Administrador.

2.3.2 HERRAMIENTAS DE PROGRAMACIÓN [10]

Las herramientas de programación utilizadas para el desarrollo del proyecto son: CooCox CoIDE y Android Studio que serán detalladas posteriormente, cabe aclarar que se ha usado un protocolo de comunicación serial UART el mismo que permite la interacción entre el microcontrolador, el sensor biométrico y la Tablet (usada como interfaz de usuario).

2.3.2.1 Uso del Entorno de Desarrollo COOCOX CoIDE-1.7.7

CooCox CoIDE es una solución de software que permite a los usuarios crear y compilar código C, para el microcontrolador STM32F4 utilizado en nuestro proyecto, y es compatible con un gran número de tablas y fichas que se puede aprovechar de acuerdo a la aplicación.

CoIDE ofrece muchas funciones para la edición de código, navegación por código, compilar, enlazar, depurar, gestionar proyectos, etc.

Características:

- De uso libre.
- IDE funcional completa.
- Plataforma de desarrollo orientado a componentes.
- Basado en Internet, la integración eficiente de los recursos de la red.
- Se integra Coos (Real-Time Operating System).
- Registros periféricos.

2.3.2.2 Requisitos para CoIDE

Requisitos del sistema: Windows XP SP3 / Windows Vista / Windows 7.

Requisitos de hardware: Adaptador de depuración: USB.

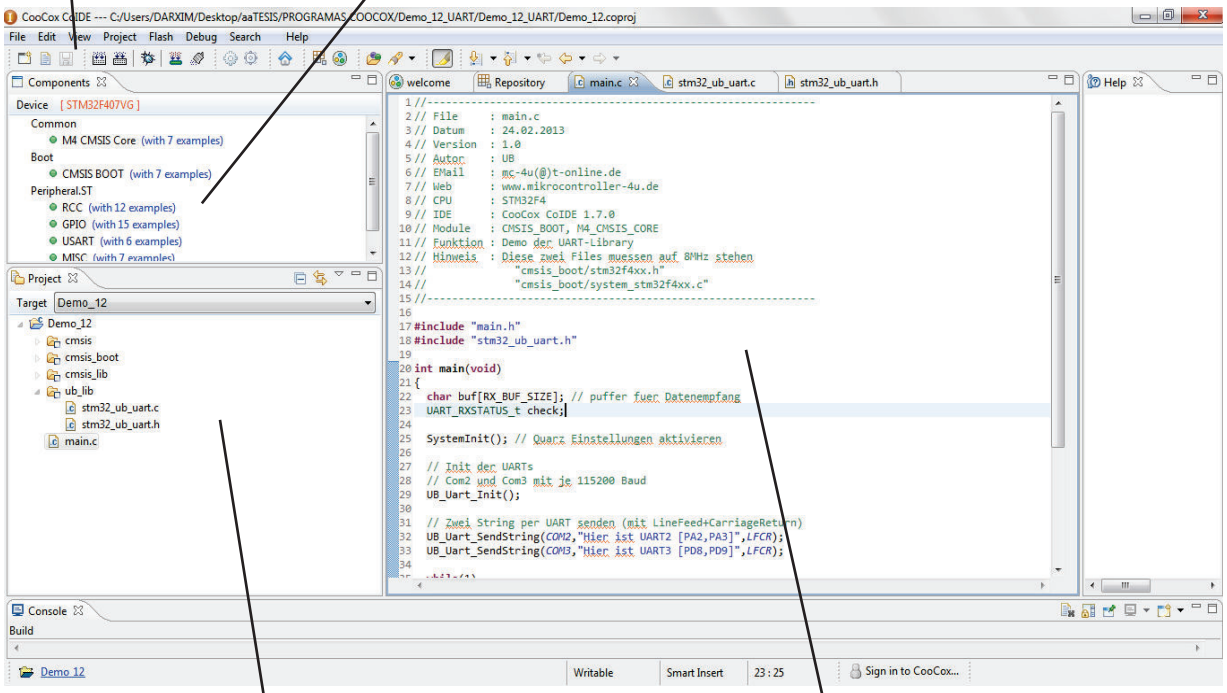
Requisitos de software:

- **GCC ARM Embedded** (Establecer compilador GCC).
- **st-link_v2_usbdriver**: Nos permite utilizar el ST-LINK incluido en la placa.
- **VCP_V1.3.1_Setup_x64:** Es un complemento para que Coocox funcione en una PC con un sistema operativo de 64 bits.
- **STM32 ST-LINK Utility_v3.3.0:** Es un depurador en circuito y programador para las familias de microcontroladores STM8 y STM32.

2.3.2.3 Ventana principal de Coocox CoIDE-1.7.7

En CooCox CoIDE, el código reutilizable se llama componente. Un componente puede ser una biblioteca, o una colección de archivos de código fuente, etc. Debe proporcionar funciones útiles, ser fácil de entender y utilizar, y lo más importante, que puede funcionar correctamente. Cada componente tiene su correspondiente fragmento de código, uso, documentación y comentarios.

Botón de Compilación



Esta vista muestra todos los archivos del sistema de archivos bajo su proyecto directorio, incluyendo archivos de origen C (* .c, * .h), archivos fuente montaje (* .s), Archivos de configuración del proyecto (build.xml) archivos de script izquierda (* .LD), Archivos de biblioteca (* .a), etc.

El editor de C / C ++ Proporciona características especializadas para la edición de archivos de C / C ++ relacionados.

2.3.2.4 Uso del Entorno de Desarrollo Android Studio

Android Studio es un entorno de desarrollo integrado dedicado para el sistema operativo Android, está diseñado para ofrecer nuevas herramientas para el desarrollo de aplicaciones. Permite ver los cambios realizados en la aplicación en tiempo real, pudiendo además probar

cómo se visualiza en diferentes dispositivos Android, como una Tablet o teléfonos inteligentes, con distintas configuraciones y resoluciones de forma simultánea.

Entre las muchas características de Android Studio se destacan sus herramientas de empaquetado y etiquetado de código para organizarlos al implementar grandes cantidades de código, sirviéndose además de un sistema drag & drop para mover los componentes a través de la interfaz de usuario.

Ventajas de Android Studio

- Ofrece un entorno de desarrollo claro y robusto.
- Facilita la prueba del funcionamiento en otros tipos de dispositivos.
- Contiene asistentes y plantillas para los elementos comunes de programación en Android.
- Cuenta con un editor con herramientas extras para agilizar el desarrollo de aplicaciones.

2.3.2.5 Requisitos de Software

- Microsoft® Windows® 8/7 / Vista / 2003 (32 o 64 bits).
- 2 GB RAM mínimo, 4 GB RAM recomendado.
- 400 MB de espacio en disco duro.
- Al menos 1 GB para Android SDK, imágenes del sistema emulador, y caches.
- 1280 x 800 resolución de pantalla mínima.
- Java Development Kit (JDK) 7.
- Opcional para emulador acelerado: procesador Intel® con soporte para Intel® VT-x, Intel® EM64T (Intel® 64) y Execute Disable (XD).
- HiSuiteSetup_v1.8.10.2006 (Instalador para tablets con Sistema Operativo Android).

2.3.2.6 Ventanas de Android Studio

- **Ventana principal**

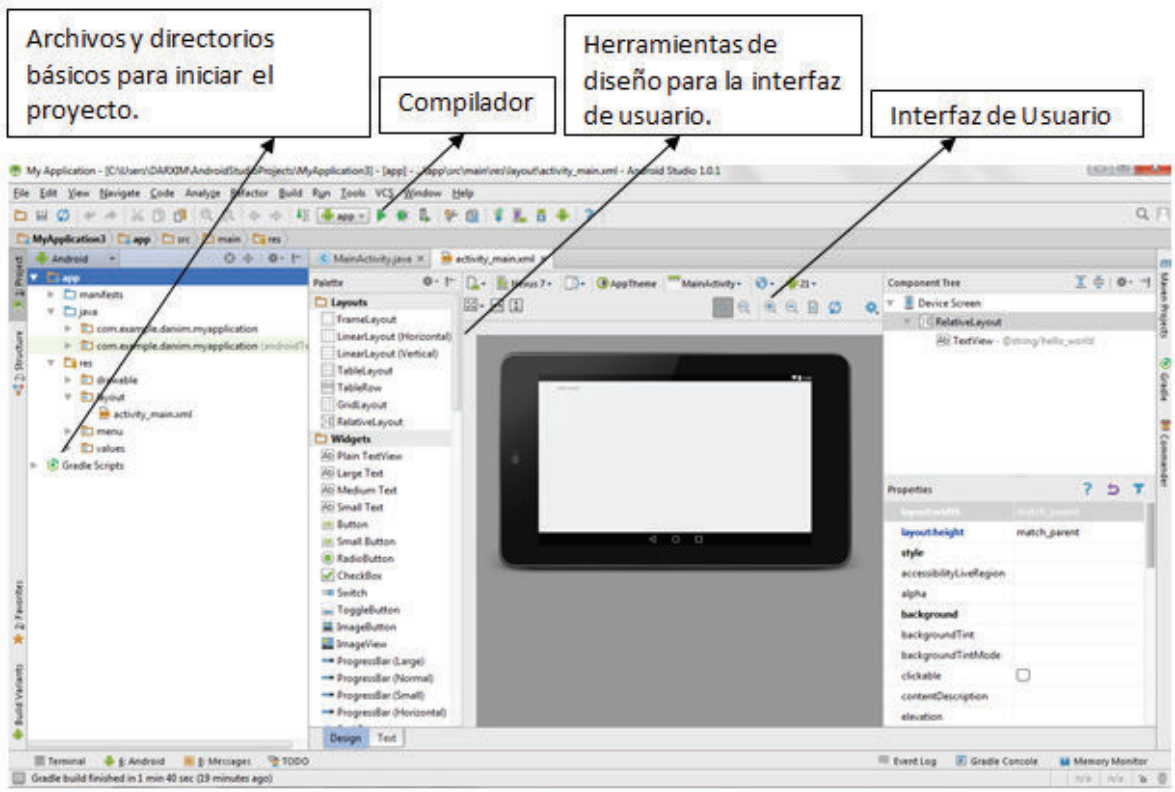


Figura 2.15: Ventana principal de Android Studio

- **Editor de código inteligente**

El núcleo de Android Studio es un editor de código inteligente, contiene todo el código fuente de la aplicación, inicialmente Android Studio creará un código básico de la pantalla (MainActivivty).

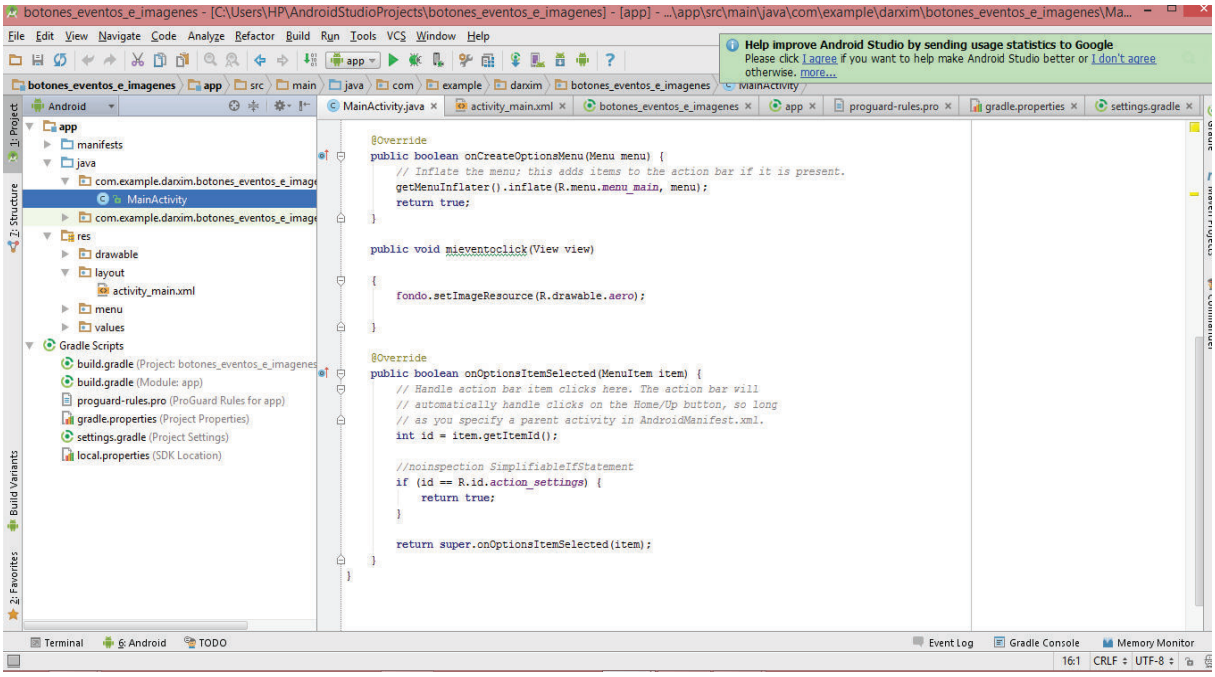


Figura 2.16: Editor de código inteligente

- **Disponibilidad de dispositivos virtuales**

Android Studio viene pre-configurado con un emulador de imagen optimizado.

El Administrador de dispositivos virtual actualizado y simplificado proporciona perfiles de dispositivo predefinidos para dispositivos Android comunes.

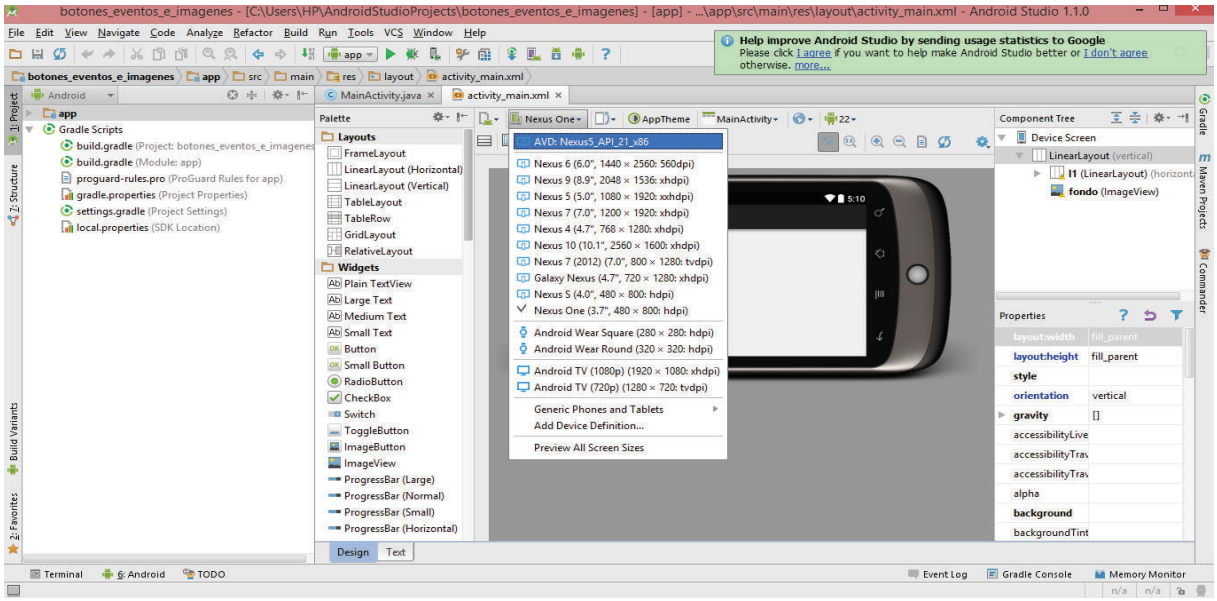


Figura 2.17: Variedad de dispositivos

2.3.3 PROGRAMACIÓN DEL MICROCONTROLADOR [11]

El programa desarrollado para controlar las funciones del microcontrolador STM32F4 contiene varias instrucciones en leguaje C destinadas a desempeñar determinadas tareas en nuestra aplicación. Este programa es grabado en el STM32F4 a través del software CooCox IDE.

2.3.3.1 Librerías utilizadas en la programación y su función

2.3.3.1.1 UART Library (STM32F4)

Con esta biblioteca se pueden utilizar los 6 UART del microcontrolador. Los UART indicados deben ser declarados en el archivo H y en el archivo C deben estar en la estructura "UART []" todos los parámetros necesarios se importan automáticamente (es decir pin asignación, velocidad de transmisión, etc.).

Se envía una función de transmisión a una cadena (string) a través de UART y se recibe una función para comprobar si una cadena se recibió a través de una UART. El remitente tiene los caracteres Identificador de fin de cadena “CarriageReturn”.

La recepción real de cada caracter por interrupción es transparente.

Funciones:

```
void UB_Uart_Init(void);                              // Inicializar los UARTs
void UB_Uart_SendByte(UART_NAME_t uart, uint16_t wert); // Para enviar un Byte
void UB_Uart_SendString(UART_NAME_t uart, char *ptr, UART_LASTBYTE_t end_cmd);
// Para enviar un string
UART_RXSTATUS_t UB_Uart_ReceiveString(UART_NAME_t uart, char *ptr); // Para
recibir un String
```

2.3.3.1.2 Button Library (STM32F4)

Esta biblioteca nos permite manejar un botón que está disponible en STM32 Discovery y puede ser utilizado por el usuario.

Las funciones de "OnPressed", "clic" y "OnReleased" pueden ser utilizadas.

Estas funciones proporcionan un "verdadero" cuando ocurrió el suceso.

Funciones:

```
Void UB_Button_Init(void);                                  // Inicializar Botones
BUTTON_STATUS_t UB_Button_Read(BUTTON_NAME_t btn_name);  //Leer el Button
bool UB_Button_OnPressed(BUTTON_NAME_t btn_name);  //true, si se pulsa el botón
bool UB_Button_OnClick(BUTTON_NAME_t btn_name);  //true, que al pulsar el botón
bool UB_Button_OnRelease(BUTTON_NAME_t btn_name);  //true, al liberar el botón
```

2.3.3.1.3 String- Library (STM32F4)

Es una biblioteca, nos ayuda en la programación C. Se trata de convertir los números a cadenas y viceversa.

En el ejemplo, se tiene prácticamente todos los tipos de conversiones para ver en el CoIDE.

Funciones:

```
Void UB_String_FloatToDezStr(float wert); //convierte un numero flotante en
String
Float UB_String_DezStringToFloat(char *ptr); //convierte un String en FLOAT
int16_t UB_String_DezStringToInt(char *ptr); //convierte un String en INT
Void UB_String_Mid(char *ptr, uint16_t start, uint16_t length);//copia una
subcadena del centro
Void UB_String_Left(char *ptr, uint16_t length);//copia la izquierda de una
cadena
Void UB_String_Right(char *ptr, uint16_t length);// copia la parte derecha de
una cadena
```

2.3.3.1.4 LED Library (STM32F4)

En el STM Discovery, hay 4 LEDs que pueden ser utilizados por el usuario. Con esta biblioteca, los LEDs se pueden cambiar de estado (ON/OFF) fácilmente.

Funciones:

```
void UB_Led_Init(void);                        // Inicialización para los LEDs
void UB_Led_On(LED_NAME_t led_name);           // Enciende un LED
void UB_Led_Off(LED_NAME_t led_name);         // Un LED apagado
void UB_Led_Toggle(LED_NAME_t led_name);      // Para alternar un LED
```

```
void UB_Led_Switch(LED_NAME_t led_name, LED_STATUS_t wert); //Para cambiar un LED
```

2.3.3.1.5 DUAL_FATFS-Library (STM32F4)

Esta biblioteca puede acceder simultáneamente en dos dispositivos con un sistema de archivos FAT (tarjeta SD y USB flash drive) los cuales se pueden operar desde el módulo de Discovery STM32F4.

Se debe tener en cuenta:

La ruta especificada para el acceso a la tarjeta SD = "0: /Test.txt"

La ruta especificada para el acceso a la memoria USB = "1: /Test.txt"

Utilización de la tarjeta SD:

Puertos usados:

```
Modo de 1-bit:
PC8: SDIO_D0 = DAT0 tarjeta SD
PC12: SDIO_CK = Tarjeta SD Reloj
PD2: SDIO_CMD = tarjeta SD CMD
Nota: la tarjeta SD Pin CD debe estar conectado a Vcc
Modo de 4 bits:
PC8: SDIO_D0 = DAT0 tarjeta SD
PC9: SDIO_D1 = DAT1 tarjeta SD
PC10: SDIO_D2 = DAT2 tarjeta SD
PC11: DAT3 SDIO_D3 = tarjeta SD / CD
PC12: SDIO_CK = Tarjeta SD Reloj
PD2: SDIO_CMD = tarjeta SD CMD
Detectar con pin:
PC0: SD_Detect-Pin (Hi = sin tarjeta SD) ya sea DMA2_STREAM3_CHANNEL4 o DMA2_STREAM6_CHANNEL4
```

Las enumeraciones:

```
typedef enum {
  FATFS_OK = 0,
  FATFS_NO_MEDIA,
  FATFS_MOUNT_ERR,
  FATFS_GETFREE_ERR,
  FATFS_UNLINK_ERR,
```

```
  FATFS_OPEN_ERR,
  FATFS_CLOSE_ERR,
  FATFS_PUTS_ERR,
  FATFS_SEEK_ERR,
  FATFS_RD_STRING_ERR,
  FATFS_RD_BLOCK_ERR,
  FATFS_WR_BLOCK_ERR,
  FATFS_EOF,
  FATFS_DISK_FULL
} FATFS_t;
```

```
typedef enum {
  F_RD = 0, // para lectura (abierta sólo si existe el archivo)
  F_WR, // abrir para escritura y adjuntar los datos (sólo si existe el archivo)
  F_WR_NEW, // abierta para la escritura (y evento. Edit) y añadir datos
  F_WR_CLEAR // abrir para escribir (Eliminar archivos viejos)
} FMODE_t;
```

Características:

```
Void UB_Fatfs_Init (void); // Init las funciones FATFS
FATFS_t UB_Fatfs_CheckMedia (MEDIA_t dev); Insertado // comprobar si medio
FATFS_t UB_Fatfs_Mount (MEDIA_t dev); // Para abrir los medios de comunicación
FATFS_t UB_Fatfs_UnMount (MEDIA_t dev); // Para cerrar el medio
FATFS_t UB_Fatfs_DelFile (const TCHAR * name); // Elimina un archivo
FATFS_t UB_Fatfs_OpenFile (FIL * fp, const TCHAR * name, FMODE_t  mode); // Para
abrir un archivo (para lectura o escritura)
FATFS_t UB_Fatfs_CloseFile (FIL * fp); // Cierra un archivo
FATFS_t UB_Fatfs_WriteString (FIL * fp, const TCHAR * tex); // Escribir una cadena
a un archivo abierto
FATFS_t UB_Fatfs_ReadString (FIL * fp, char * tex, int len); // Lee una cadena
desde un archivo abierto
uint32_t UB_Fatfs_FileSize (FIL * fp); // Leer Del Tamaño del archivo
FATFS_t  UB_Fatfs_ReadBlock  (FIL  *  fp,  sin  firmar  char  *  buf,  uint32_t  len,
uint32_t * read); // Leer archivo en bytes
FATFS_t UB_Fatfs_WriteBlock (FIL * fp, unsigned char * buf, uint32_t len, uint32_t
* write); // Escribir Archivo byte a byte
```

Utilización del USB flash drive:

Puertos usados

```
PA9 -> USB_OTG_VBUS
PA10 -> USB_OTG_ID
```

```
PA11 -> USB_OTG_DM
PA12 -> USB_OTG_DP
PC0 -> USB_VBUS_Enable
```

Las enumeraciones:

```
typedef enum {
  USB_MSC_HOST_NO_INIT = 0, // interfaz USB no se ha inicializado
  USB_MSC_DEV_DETACHED, // conectado ningún dispositivo
  USB_MSC_SPEED_ERROR, // No se admite Speed USB
  USB_MSC_DEV_NOT_SUPPORTED, // Dispositivo no soportado
  USB_MSC_DEV_WRITE_PROTECT, // El dispositivo está protegido contra escritura
  USB_MSC_OVER_CURRENT Reconocido, // sobrecorriente
  USB_MSC_DEV_CONNECTED conectado // dispositivo listo
} USB_MSC_HOST_STATUS_t;
```

Características:

```
Void UB_USB_MSC_HOST_Init (void); // Init al host USBUSB_MSC_HOST_STATUS_t
UB_USB_MSC_HOST_Do (void); // Para comprobar el estado del dispositivo
```

2.3.3.1.6 USB_MSC_HOST-Library (STM32F4)

Con esta librería USB-Sticks, el puerto USB OTG puede ser operado desde el módulo Discovery.

La biblioteca tiene dos funciones:

1) "Función “Init”

La función debe ser llamada una vez, al inicio.

2) La "función “HOST_Do”

Esta función debe ser periódica while (1). Este estado debe ser evaluado cuando se ha detectado una USB a la que se puede acceder por FATFS it.

Es necesario un adaptador USB con conector micro USB.

Funciones:

```
Void UB_USB_MSC_HOST_Init(void);              //Inicializar al Host USB
```

```
USB_MSC_HOST_STATUS_t UB_USB_MSC_HOST_Do(void); //Para comprobar el estado del
dispositivo
```

2.3.3.1.7 AV_ZFM20X- Library (STM32F4)

Esta librería le permite al microcontrolador enviar y recibir datos desde y hacia el biométrico.

Estos datos representan comandos de control o ejecutan una determinada función (enrolar, identificar, buscar o borrar usuarios).

Funciones:

```
UB_Uart_SendArray(COM2,char,dirección);   //Para enviar un Array
UB_Uart_ReciveArray(COM2,char);           // Para recibir un Array
```

2.3.4 PROGRAMACIÓN DE LA TABLET

El Software utilizado para programar nuestra interfaz de usuario en la Tablet es Android Studio.

El programa asignado contiene varias instrucciones en leguaje JAVA destinadas a desempeñar determinadas funciones para la ejecución de nuestra aplicación.

2.3.4.1 Componentes básicos de Android y su función

2.3.4.1.1 Activity

Una actividad o activity es el componente básico de la interfaz gráfica de la aplicación, permite crear componentes que interactúan con el usuario. Cada actividad está asociada con una vista o view la cual constituye un conjunto de elementos gráficos que se presentan al usuario para que interactúe con el sistema. Por ejemplo, bloques de edición, etiquetas o botones, entre otros.

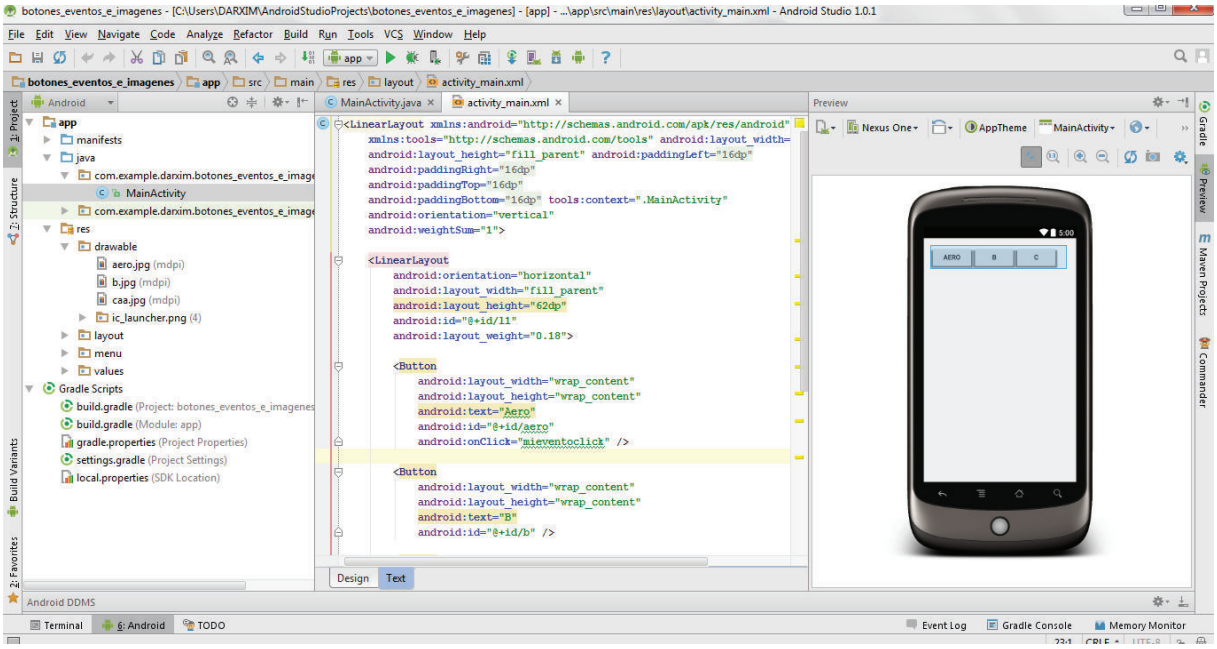


Figura 2.18: Interfaz gráfica

2.3.4.1.2 Intent

Una intención o intent es un componente básico de Android que permite el envío de mensajes entre componentes entendiéndose por componentes:

Las activities (componentes de Interfaz gráfica), services (código que se ejecuta en segundo plano), receivers (código que responde a un mensaje de transmisión), proveedores de contenido (código que abstrae los datos).

Los intents son mecanismos para llamar a aplicaciones externas a la nuestra, emitir eventos a los que otras aplicaciones puedan responder.

Código:

```
public class MiActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.MiActivity);
    }
}
```

2.3.4.1.3 View

Son componentes con los que se construye la interfaz gráfica de la aplicación. Por ejemplo: cuadros de texto, botones, listas plegables o imágenes. Existe la posibilidad de extender la funcionalidad de dichos controles o crear controles personalizados.

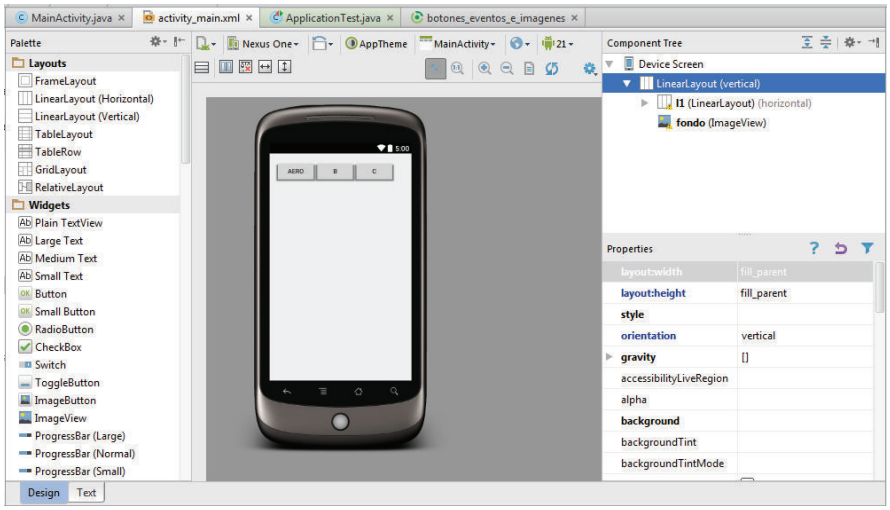


Figura 2.19: Componentes de la interfaz gráfica

2.3.4.1.4 Service

Son componentes que no cuentan con una interfaz gráfica de usuario pero se ejecutan en segundo plano, los Servicios pueden ejecutar cualquier tipo de tarea, por ejemplo: actualizar datos, mostrar notificaciones.

2.3.4.1.5 Broadcast Receiver

Un Broadcast Receiver o receptor de notificaciones es un componente utilizado para detectar y reaccionar ante determinados eventos generados por el sistema por ejemplo: batería baja, tarjeta SD insertada.

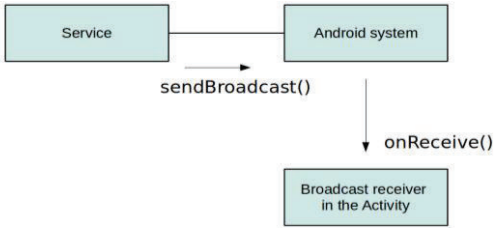


Figura 2.20: Ejemplo de Broadcast Receiver[18]

[18] http://www.vogella.com/tutorials/AndroidServices

2.3.4.1.6 Layouts

Son elementos no visuales utilizados para controlar la distribución, dimensión y posición de los controles que se insertan en su interior.

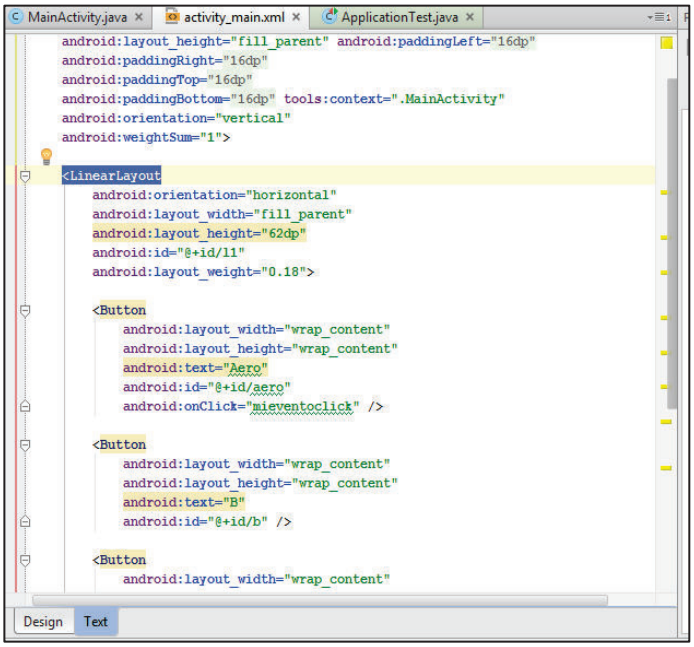


Figura 2.21: Ejemplo de Layouts

2.3.5 SENTENCIAS UTILIZADAS EN LA PROGRAMACIÓN [12]

2.3.5.1 Sentencia FLOAT

Sirve para declarar variables fraccionarias o de punto flotante por ejemplo: 2.83 o 34.543.

2.3.5.2 Sentencia CONST

Sirve como medida de seguridad, representa una constante, es decir una variable que nunca se debe modificar.

2.3.5.3 Sentencia INT

Son tipos de datos que indican un número entero y por lo tanto no pueden ser nombres de variables.

2.3.5.4 Sentencia VOID

Sirve para declarar un proceso que no tiene un valor de retorno, también se utiliza para definir punteros genéricos. Estos tipos de punteros no están relacionados con ningún tipo de dato específico, sino solamente se utilizan para almacenar posiciones de memoria.

2.3.5.5 Sentencia BOOL

Declara un tipo especial de variable denominada Booleana, que solo puede tener dos valores: cierto o falso, a estas variables también se les denomina variables lógicas.

2.3.5.6 Sentencia FOR

Es una variable de control que se inicializa en cero, después se entrega la condición de control que será falsa durante la ejecución para completar el ciclo; y por último tenemos el incrementador en una unidad. Tiene la siguiente sintaxis: for ([variable contadora=variable de inicio]; [condición que se debe cumplir para finalizar]; [incremento de la variable contadora]) {lo que se repite tantas veces como tarde la variable en determinar el límite fijado}.

2.3.5.7 Sentencia IF

Es un condicional que pide que elija entre dos opciones y cada opción tiene una consecuencia. Si cumple con las condiciones que nosotros especificamos en el programa, éste continuará ejecutándose.

2.3.5.8 Sentencia WHILE

Es un ciclo, es decir repite una sentencia o secuencia de sentencias un número de veces. El ciclo While tiene una condición, una expresión lógica que controla la secuencia de repetición.

2.3.5.9 Sentencia ELSE

Sirve como una condicional junto con la instrucción If, es decir que es una opción alternativa que cumple con otras condiciones diferentes que la sentencia If no tiene. Esta se ejecuta siempre y cuando la declaración If sea falsa.

2.3.5.10 Sentencia RETURN

Esta sentencia indica al compilador que tiene que regresar al valor de retorno que se le indique en una determinada función.

2.3.5.11 Sentencia BREAK

Es una sentencia de finalización, finaliza la ejecución de un bucle sin finalizar el resto de las sentencias asociadas al mismo.

2.3.5.12 Sentencia CHAR

Son tipos de datos que indican una cadena de caracteres, por ejemplo: nombres de variables.

2.3.5.13 Sentencia SWITCH CASE

Se utiliza para agilizar la toma de decisiones múltiples, es decir nos permite elegir entre muchas opciones.

2.4 ALIMENTACIÓN DEL SISTEMA

El sistema se encuentra alimentado a través de la Tablet, la misma que cuenta con un cargador el cual deberá permanecer conectado siempre al toma corriente, de esta manera la Tablet será la encargada de alimentar a todo el conjunto de dispositivos interconectados.

En caso de que no se cuente con suministro eléctrico, se dispone de una alimentación redundante que es la batería de la Tablet, esta es recargable y tiene una duración de 120hrs. en Standby.

2.5 IMPLEMENTACIÓN

Esta sección se refiere a la interconexión física de los elementos electrónicos que forman parte del sistema para posteriormente ubicarlos en un módulo metálico el cual cumple la función de proteger la circuitería electrónica de las condiciones del medio externo como humedad y polvo.

El dispositivo implementado consta de los siguientes elementos:

1) Placa de desarrollo STM 32F4 Discovery Review

2) Sensor Biométrico ZFM-20

3) Convertidor USB- SERIE FT232RL FTDI

4) Cable adaptador USB Hembra a Mini USB

5) Cable adaptador USB Hembra a Micro USB

6) Circuito auxiliar

7) Cargador

8) Tablet

9) Dispositivo de almacenamiento Flash USB (accesorio).

10) Tarjeta de almacenamiento micro SD.

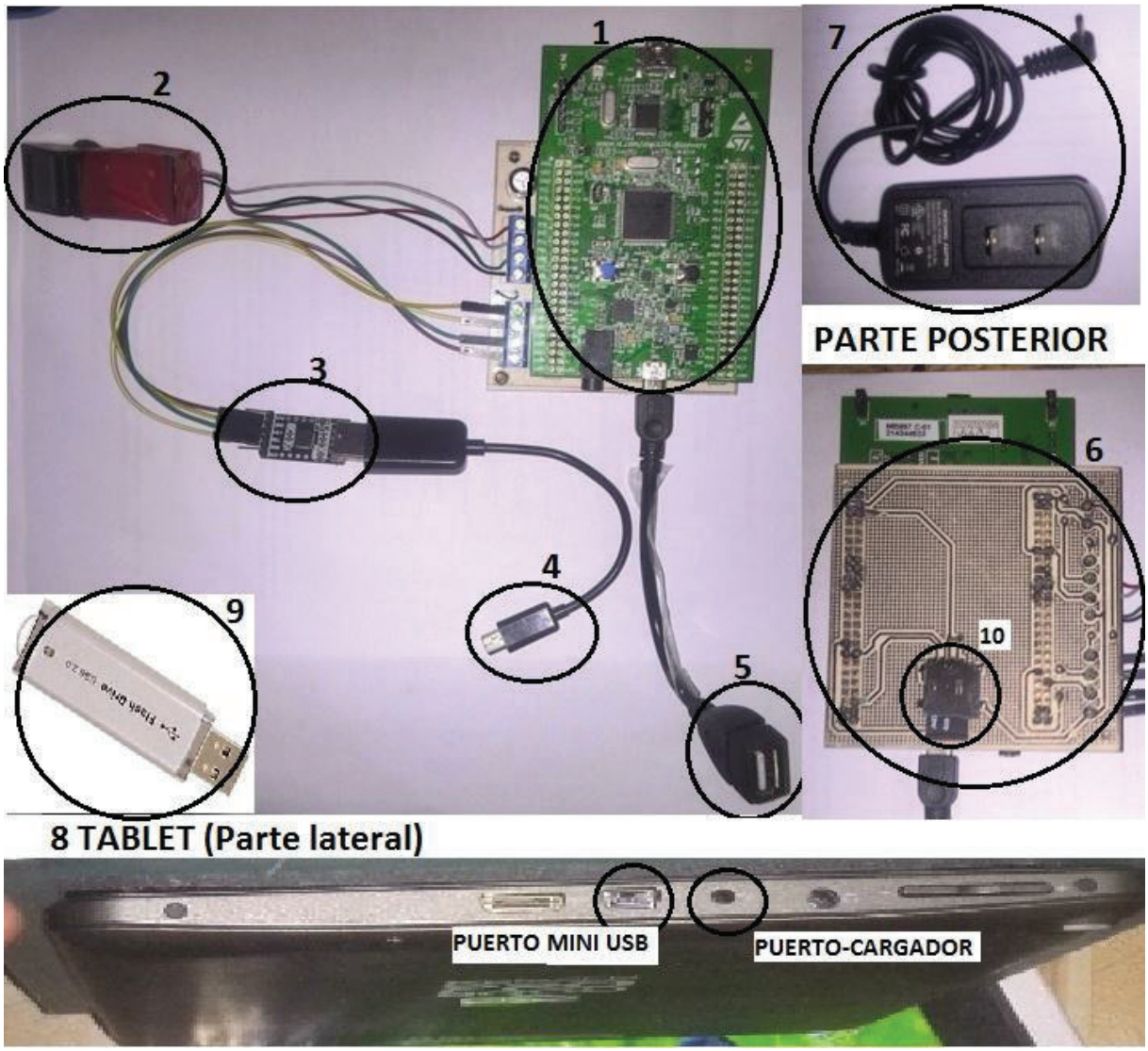


Figura 2.22: Elementos y accesorios del Dispositivo.

El elemento principal del sistema es la placa de desarrollo STM 32F4 Discovery Review a la cual se conectan los otros elementos. La interconexión de dichos elementos es relativamente sencilla ya que los pines de conexión utilizados están claramente definidos en las hojas técnicas del STM32F4.

El circuito auxiliar simplemente es un circuito impreso cuyas pistas enlazan los pines de conexión utilizados hacia borneras para facilitar la implementación del sistema.

A continuación se muestra el montaje del sistema en el módulo metálico.

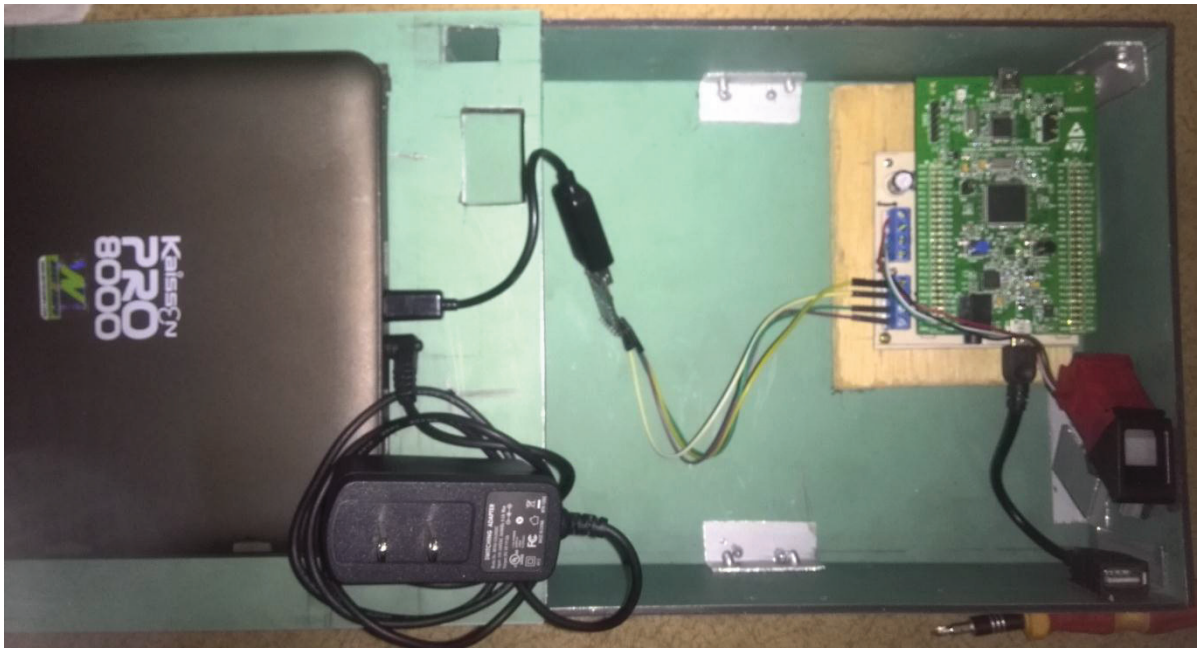


Figura 2.23: Implementación del sistema en el módulo metálico

2.5.1 PRUEBAS Y RESULTADOS EN LA IMPLEMENTACIÓN DEL SISTEMA

En esta sección se realizan pruebas de las funciones del sistema para verificar su correcto funcionamiento y corregir posibles fallas.

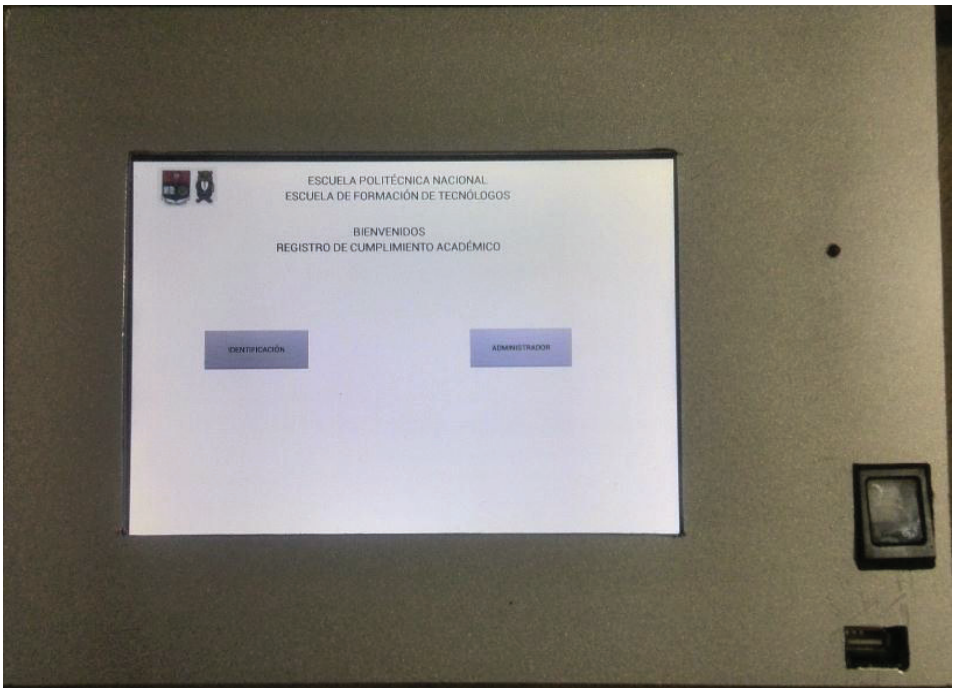


Figura 2.24: Pantalla principal.

2.5.1.1 Prueba de ingreso administrador "Clave"

En esta prueba se verifica que se puede ingresar al modo ADMINISTRADOR únicamente si se cuenta con la clave administrador (10031723).

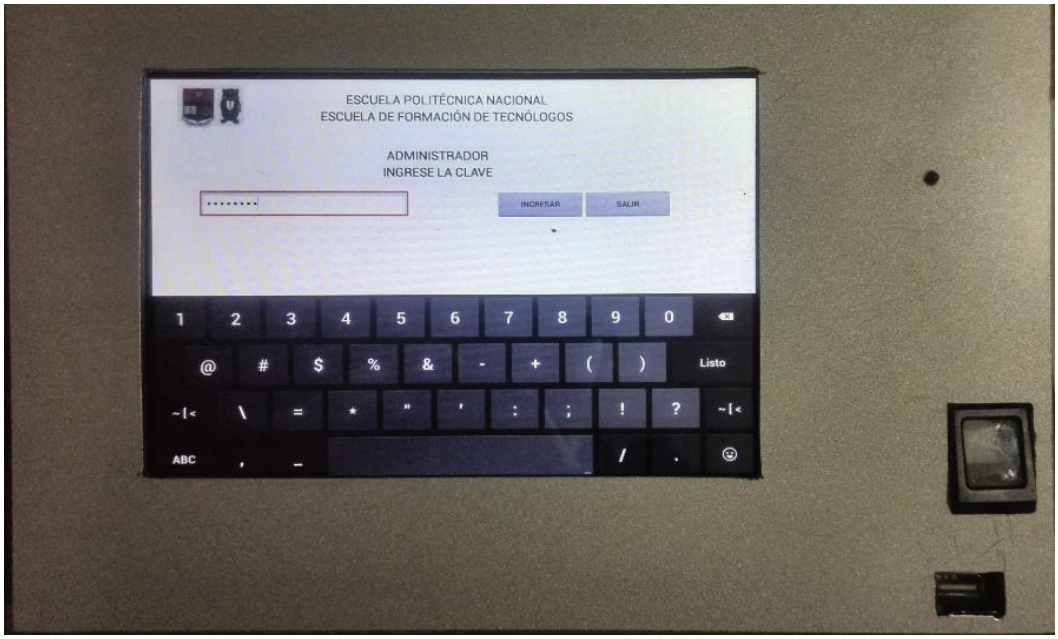


Figura 2.25: Pantalla INGRESAR CLAVE.

Al ingresar la clave correcta el sistema muestra el mensaje "CLAVE CORRECTA" y nos permite ingresar al menú de Administrador.

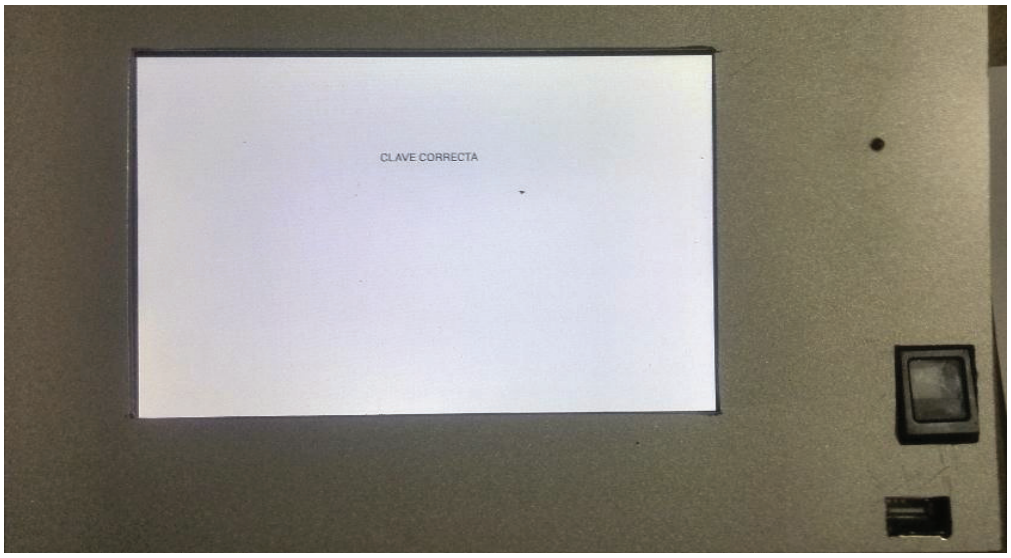


Figura 2.26: Pantalla CLAVE CORRECTA.

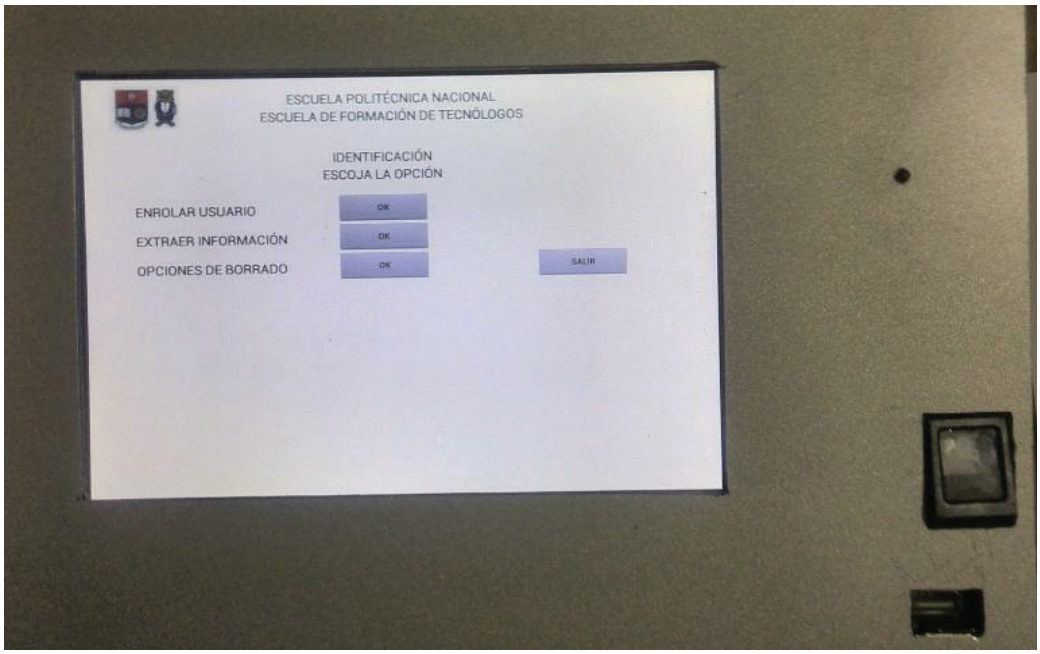


Figura 2.27: Pantalla MENÚ ADMININSTRADOR.

Si se ingresa una clave incorrecta el sistema muestra el mensaje "CLAVE INCORRECTA" para después regresar a la pantalla principal.

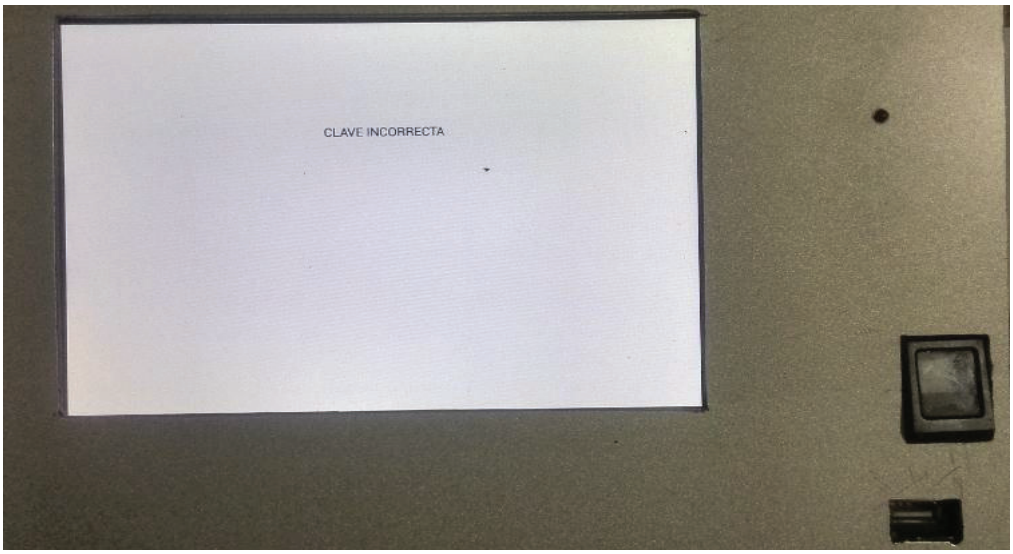


Figura 2.28: Pantalla CLAVE INCORRECTA.

2.5.1.2 Prueba de enrolamiento de usuario

Para enrolar un usuario (se oprime el botón ENROLAR USUARIO en la pantalla MENÚ ADMINISTRADOR) el sistema pide escribir los datos del usuario, una vez hecho esto se debe presionar la tecla guardar y el sistema nos dirigirá a la pantalla de COLOCAR DEDO para luego guardar los datos en la memoria interna del sistema el proceso se confirma cuando se muestra la pantalla ENROLADO EXITOSO.

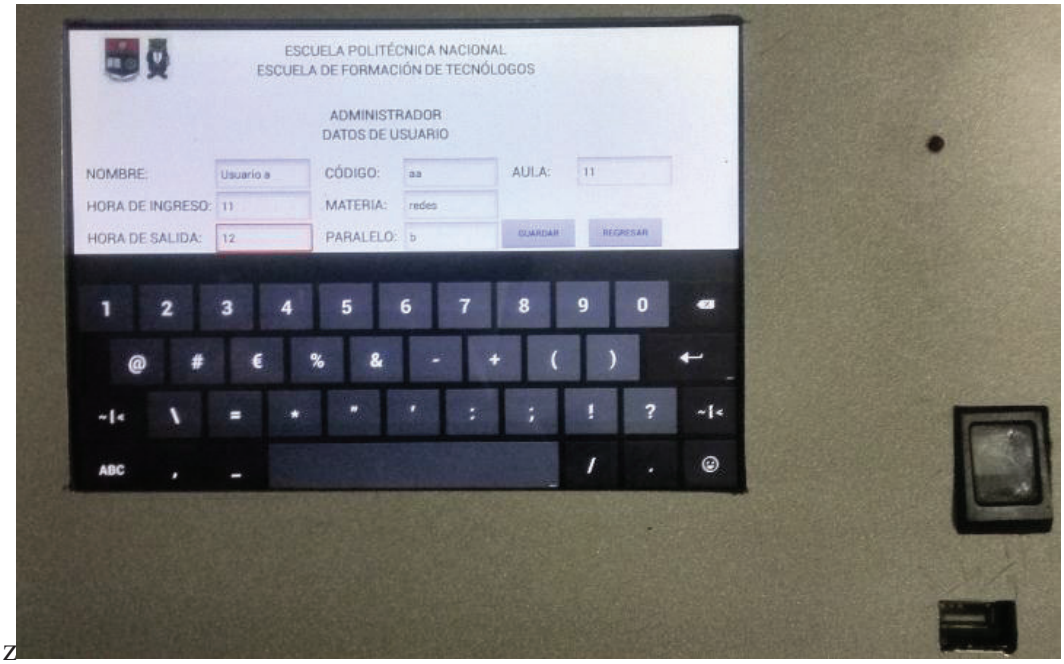


Figura 2.29: Pantalla DATOS DE USUARIO.

Figura 2.30: Pantalla COLOCAR DEDO.

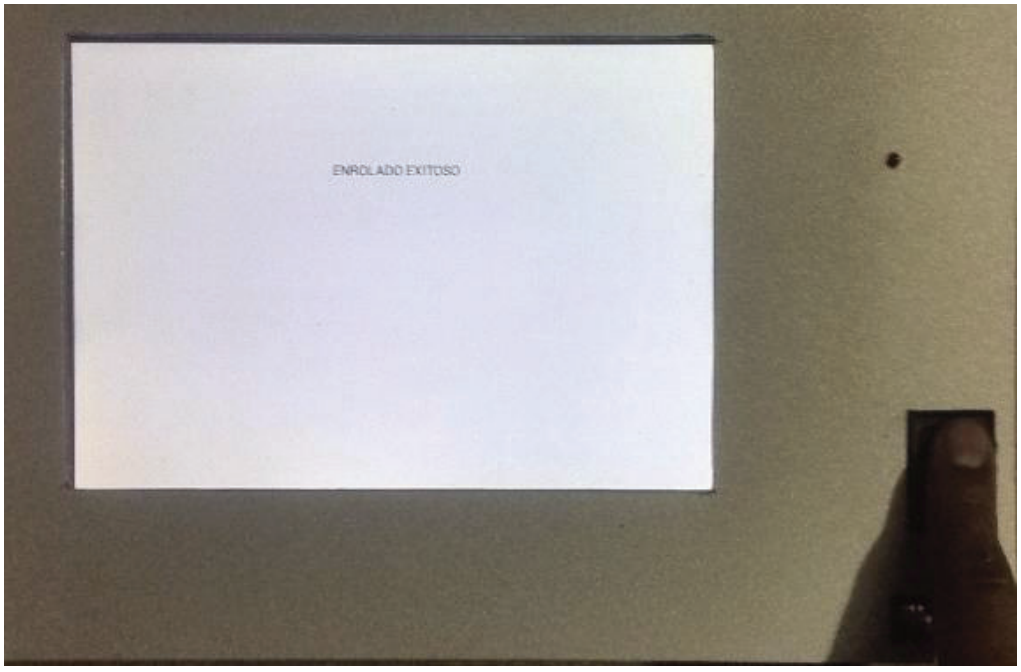


Figura 2.31: Pantalla ENROLADO EXITOSO.

Si el dedo no se coloca en el biométrico el sistema debe mostrar el mensaje "HUELLA INCORRECTA" y regresará a la pantalla DATOS DE USUARIO.

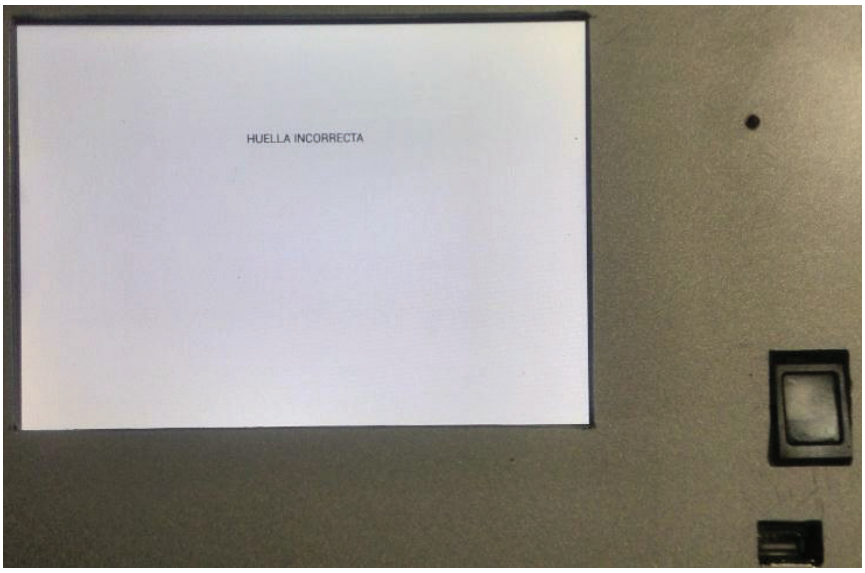


Figura 2.32: Pantalla HUELLA INCORRECTA.

2.5.1.3 Prueba de Identificación y Registro

En esta prueba se verifica la identidad de usuario y se le permite escribir un mensaje de texto (tema dictado). Primero el sistema le pedirá colocar su dedo en el lector de huellas si la huella ingresada consta con la huella de un usuario anteriormente enrolado, el sistema le pedirá ingresar el texto para después guardar el registro, esto se confirma cuando se muestra el mensaje "REGISTRO EXITOSO", si la huella ingresada no coincide con la huella de un usuario anteriormente enrolado se muestra el mensaje " USUARIO NO ADMITIDO " y el sistema regresará a la pantalla principal.

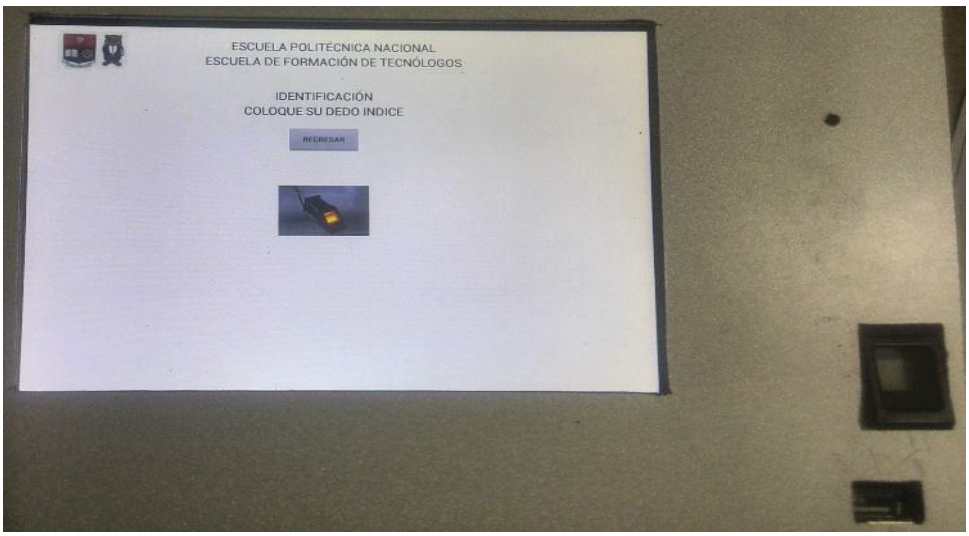


Figura 2.33: Pantalla IDENTIFICACIÓN COLOCAR DEDO.

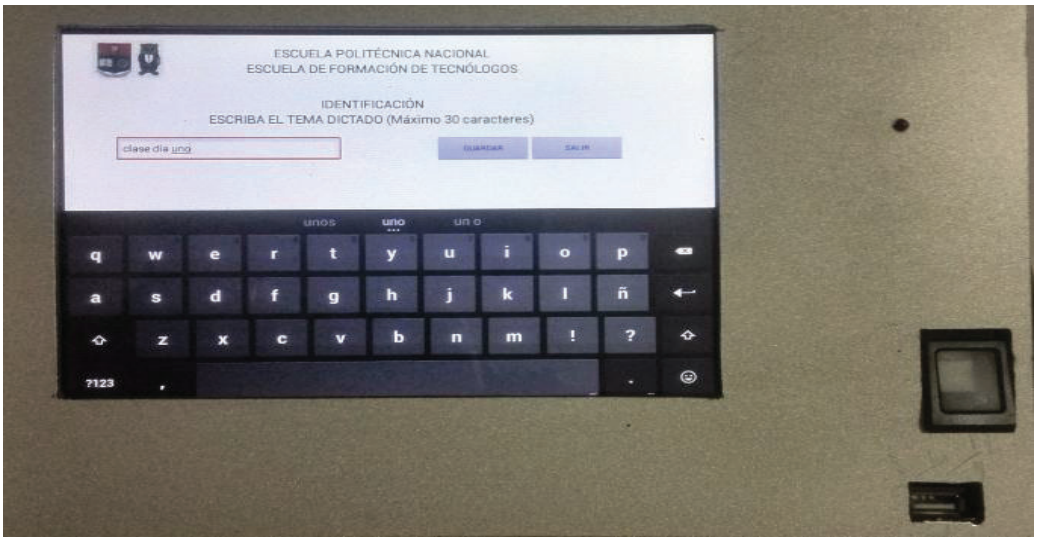


Figura 2.34: Pantalla TEMA DICTADO

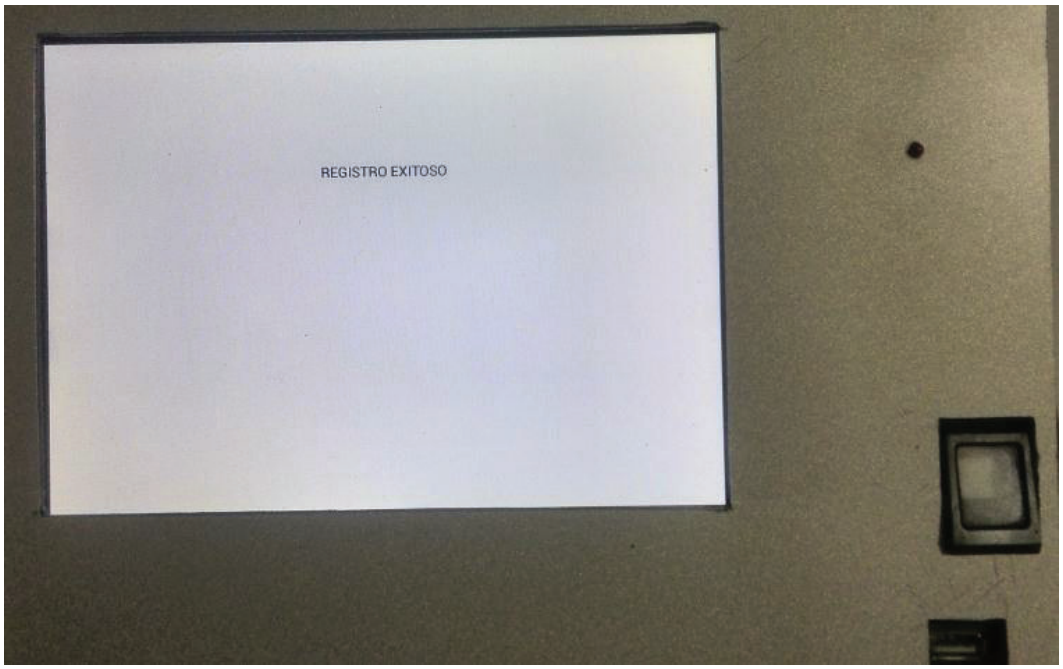


Figura 2.35: Pantalla REGISTRO EXITOSO.

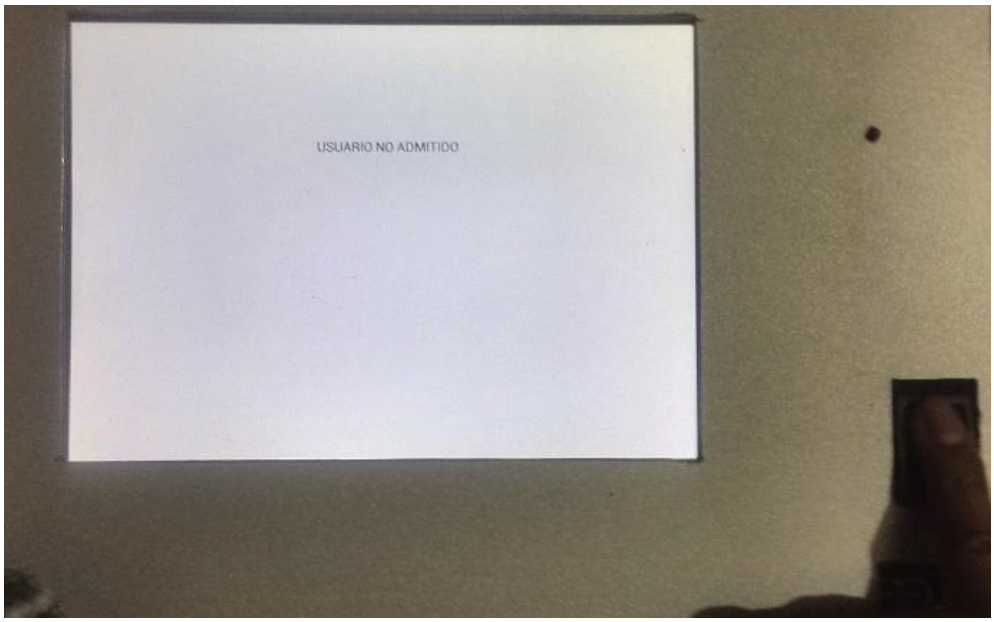


Figura 2.36: Pantalla USUARIO NO ADMITIDO.

2.5.1.4 Prueba de Extracción de Información

Para esta prueba el sistema únicamente nos pide colocar una memoria USB la cual se debe ingresar en la ranura dedicada para este propósito si es así se muestra el mensaje "INFORMACIÓN EXTRAÍDA" y regresará a la pantalla MENÚ ADMINISTRADOR.

Si no se coloca una memoria USB el sistema simplemente regresará a la pantalla MENÚ ADMINISTRADOR.

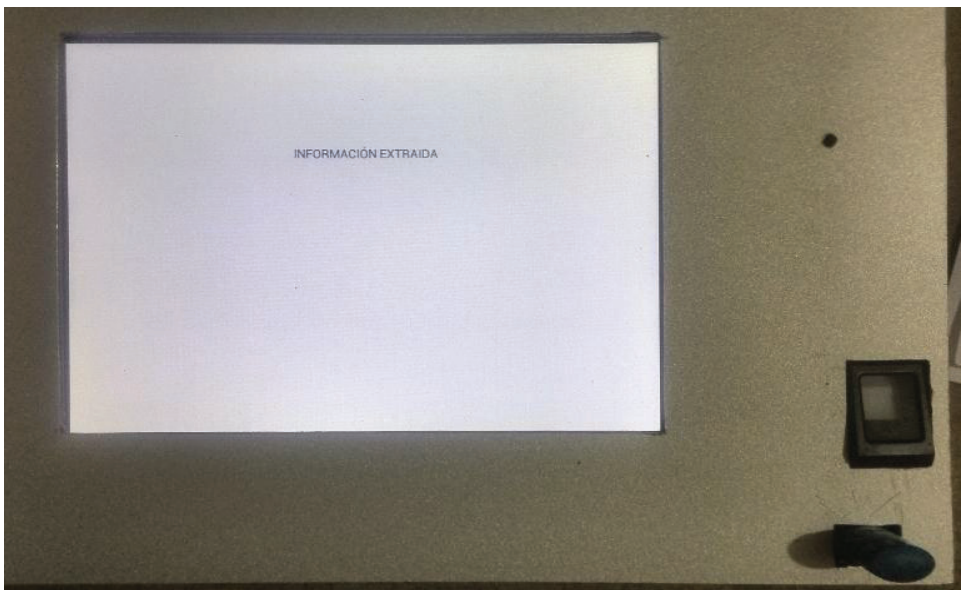


Figura 2.37: Pantalla INFORMACIÓN EXTRAÍDA.

El sistema extrae la información en forma de dos archivos de texto (REGISTRO y USUARIOS) así que el siguiente paso es verificar el proceso poniendo la memoria USB en un computador de propósito general para leer los archivos generados.

En la figura 2.38 se puede apreciar el archivo USUARIOS en donde constan los datos del usuario enrolado anteriormente (ver figura 2.29).

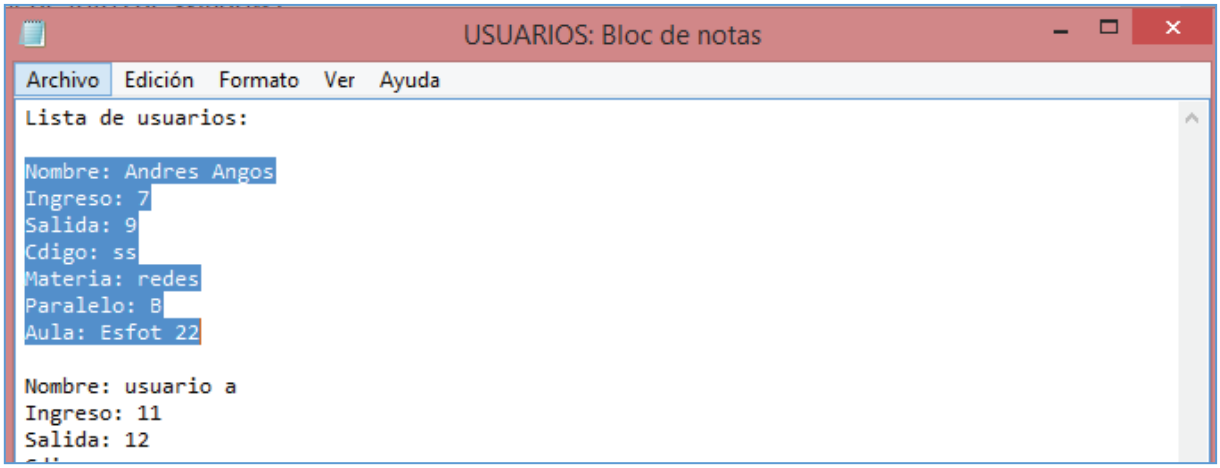


Figura 2.38: Archivo USUARIOS

En la figura 2.39 se observa el archivo REGISTRO, en el archivo podemos apreciar el registro que hizo anteriormente el usuario en el que consta la fecha, hora, tema dictado y el nombre del usuario.

Figura 2.39: Archivo USUARIOS

Nota: Los archivos tipo texto ocupan una mínima cantidad de memoria, en la figura 2.40 se observa que el tamaño de los archivos es de 1KB, entonces, como la memoria que tiene el sistema para guardar dichos archivos es de 2GB concluimos que prácticamente se puede guardar una gran cantidad de información en el sistema.

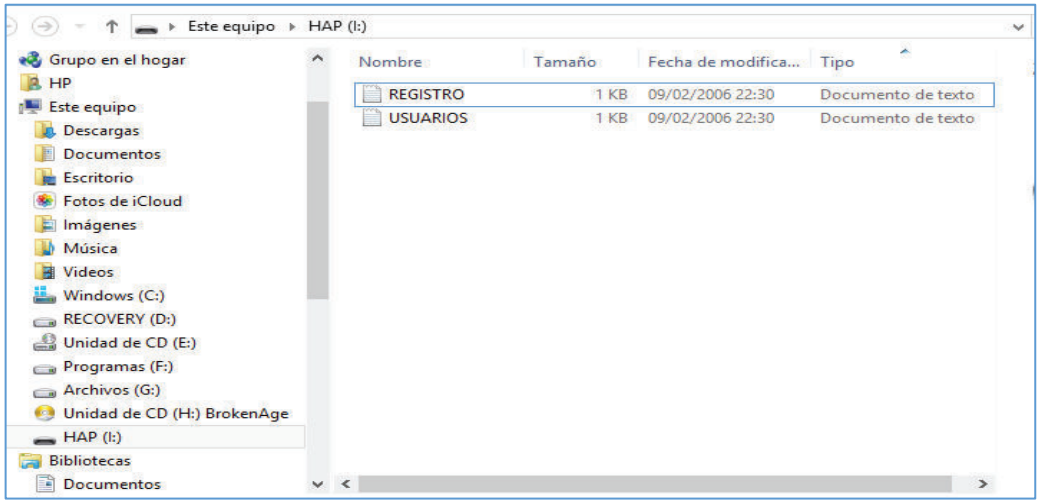


Figura 2.40: Tamaño de los archivos.

2.5.1.5 Prueba de Opciones de Borrado

Esta prueba tiene como objetivo verificar que se puede borrar la información existente en el sistema como usuarios o el mismo registro de actividades, se usará la opción borrar todos los usuarios (botón BORRAR T)

En la figura 2.41 se observa la lista de usuarios enrolados la cual vamos a borrar del sistema.

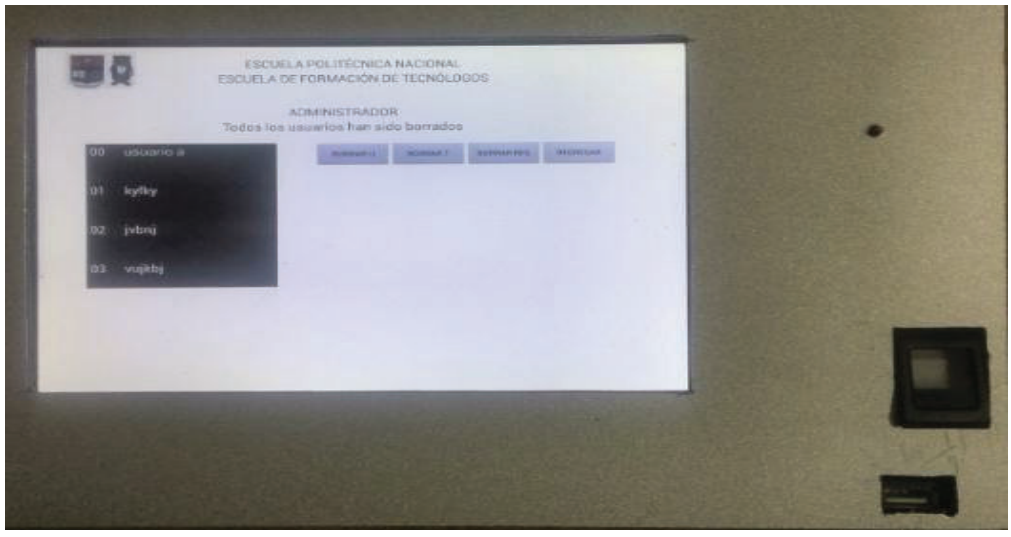


Figura 2.41: Pantalla OPCIONES DE BORRADO.

Para comprobar que el sistema ha borrado a todos los usuarios se presionará el botón regresar y luego se ingresa de nuevo a OPCIONES DE BORRADO, entonces aparecerá la lista de usuarios completamente vacía (figura 2.42).

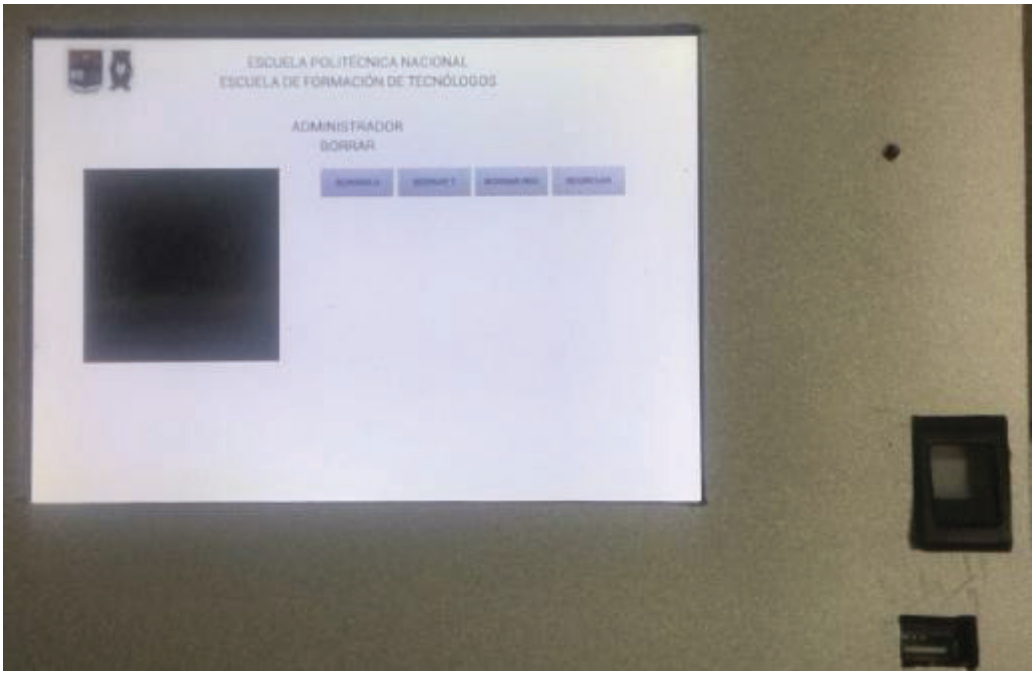


Figura 2.42: Lista de usuarios borrada.

CAPÍTULO 3.

MANUAL DEL USUARIO

3.1 PRINCIPALES PARTES Y ACCESORIOS DEL DISPOSITIVO

El dispositivo implementado consta de las siguientes partes accesibles al usuario explicadas en la siguiente figura:

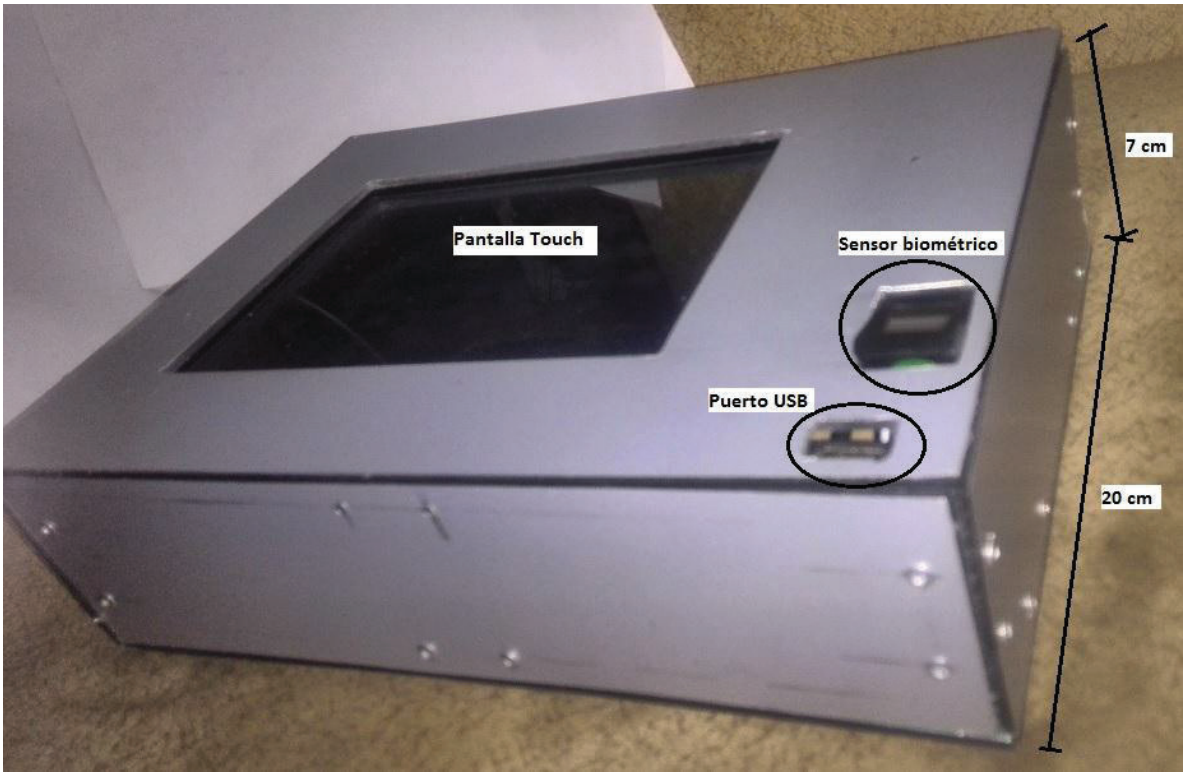


Figura 3.1: Principales partes y accesorios del dispositivo

3.2 PROCESO DE INGRESO DE NUEVOS USUARIOS AL SISTEMA (ENROLAMIENTO).

Este proceso consiste en ingresar a los docentes al sistema, se lo deberá realizar al inicio de cada semestre o cuando el administrador requiera incorporar a un nuevo docente no registrado en la base de datos. Para lo cual se deberá seguir los pasos detallados a continuación:

1.- El sistema permanecerá en estado activo continuamente, mostrando la pantalla principal como se muestra en la figura 3.2:

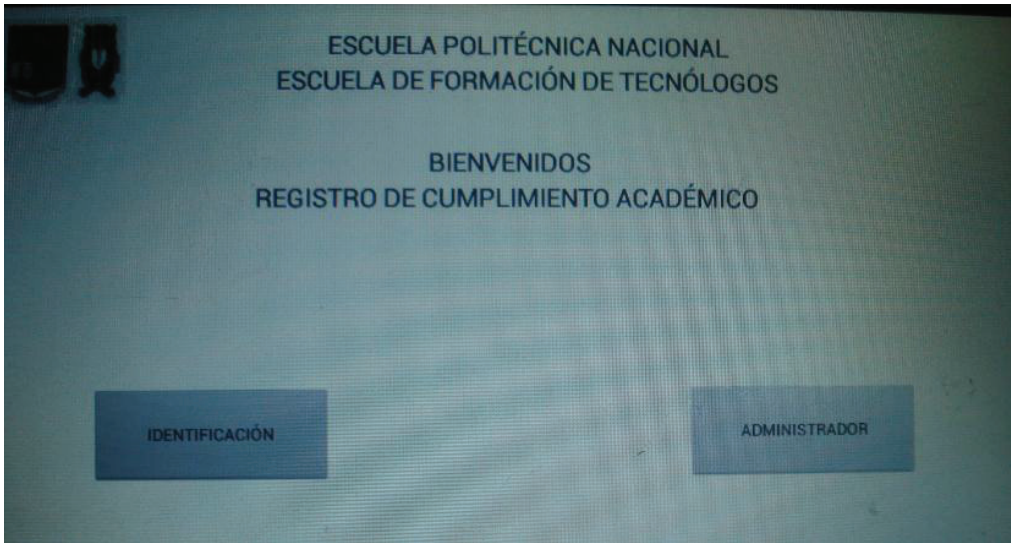


Figura 3.2: Pantalla de inicio

2.- Seguidamente se elegirá en la pantalla la opción de ADMINISTRADOR, a continuación el sistema pedirá que se ingrese la Clave de administrador (10031723), como lo indica la figura 3.3:

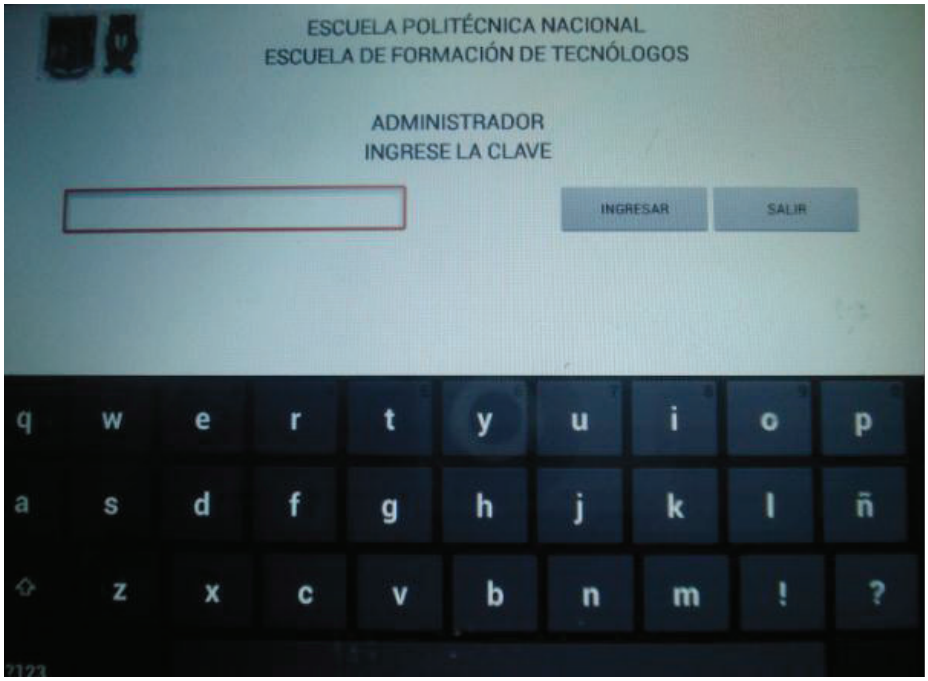


Figura 3.3: Solicitud de la clave de Administrador

3.- Una vez que se haya ingresado correctamente la clave, se desplegará en el menú las siguientes opciones: Enrolar Usuario, Extraer Información y Opciones de Borrado como lo indica la figura 3.4. Se elegirá la opción: ENROLAR USUARIO.

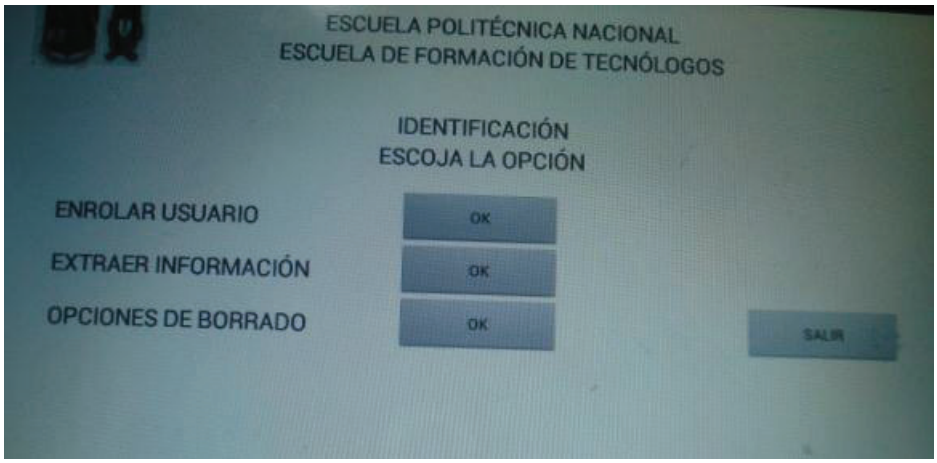


Figura 3.4: Menú del Administrador

4.- A continuación aparecerá en la pantalla de la Tablet casilleros que deberán ser llenados con los datos del usuario a registrar como se muestra en la figura 3.5.

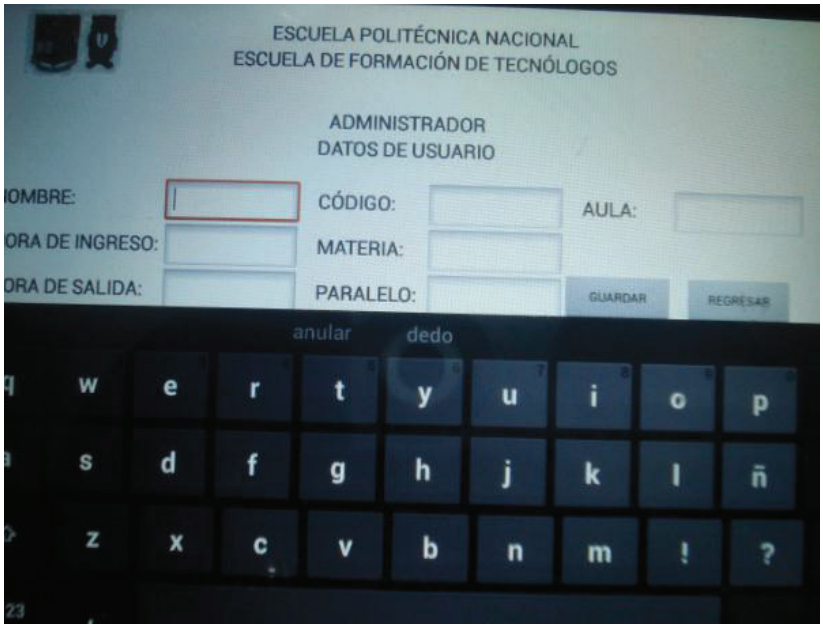


Figura 3.5: Datos del usuario

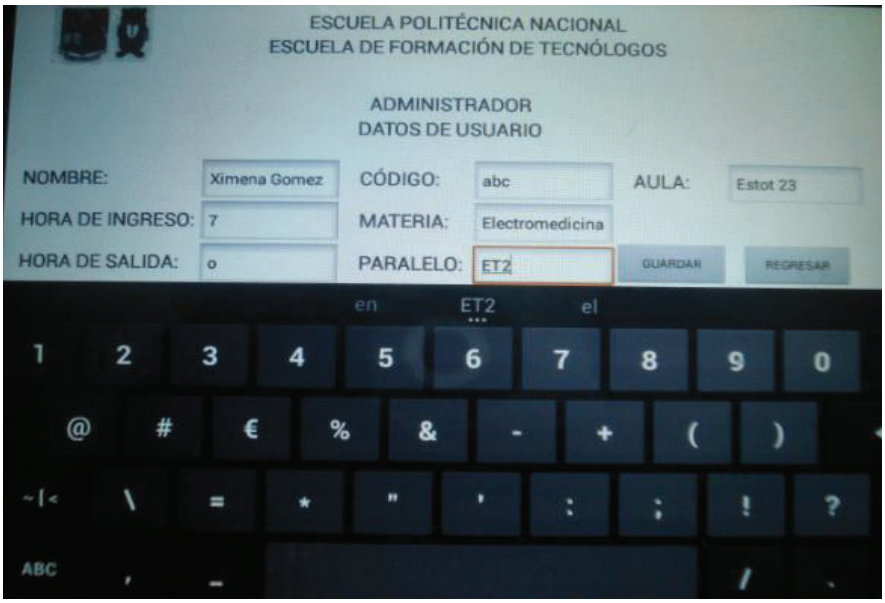


Figura 3.6: Casillas llenas con los datos del usuario

5.- Después que todos los datos de usuario han sido ingresados, se presionará el botón GUARDAR.

6.- Para continuar con el proceso de enrolamiento el sistema mostrará el siguiente mensaje: "COLOCAR DEDO DEL USUARIO"; Para lo cual el usuario debe colocar su dedo (índice de preferencia) en el sensor biométrico.

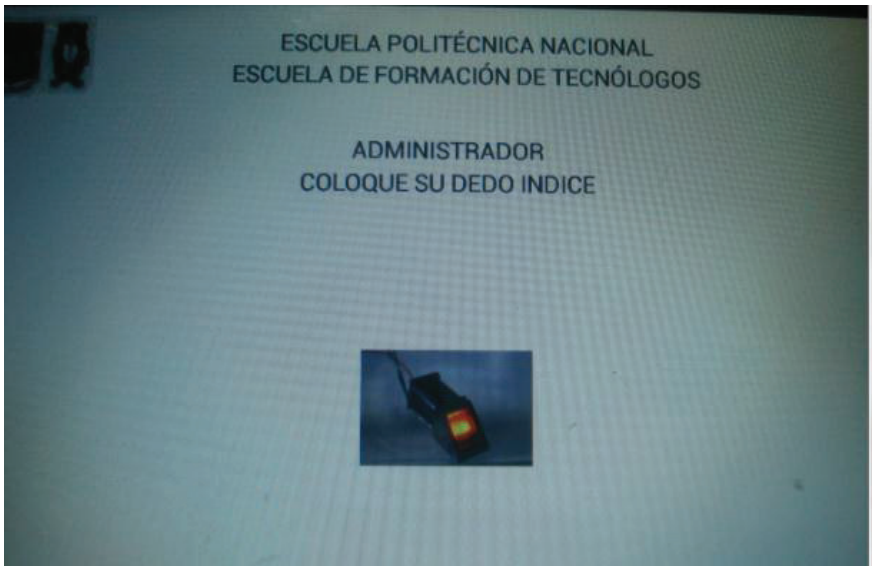


Figura 3.7: Mensaje de colocación del dedo en el Sensor Biométrico

Nota: El sistema cuenta con 10 parpadeos medidos en tiempo para que el usuario coloque su dedo en el biométrico caso contrario se desplegará un mensaje: "HUELLA INCORRECTA" y volverá a pedir que se ingresen nuevamente los datos de usuario.

7.- Si el proceso de enrolamiento se ha llevado a cabo exitosamente, el sistema mostrará el mensaje: "ENROLAMIENTO EXITOSO".

3.3 PROCESO DE DESCARGA DE INFORMACIÓN

Este proceso será realizado únicamente por el administrador, consiste en la extracción de la información almacenada en la base de datos del sistema para lo cual se debe seguir los siguientes pasos:

1.- Nuevamente se elegirá la opción de ADMINISTRADOR, a continuación el sistema pedirá que se ingrese la Clave de administrador (10031723).

2.- Se desplegará en el menú las siguientes opciones: Enrolar Usuario, Extraer Información y Opciones de Borrado. Se elegirá la opción EXTRAER INFORMACIÓN

3.- Como se muestra en la figura 3.8. El sistema presenta en la pantalla el mensaje "INSERTAR MEMORIA USB.

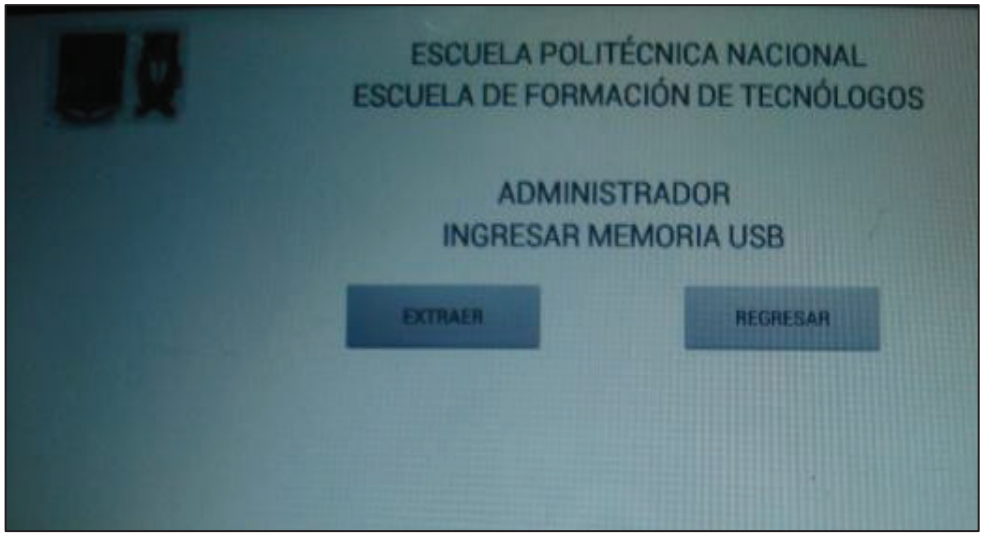


Figura 3.8: Mensaje de insertar la memoria USB

4.- A continuación se digitará el botón EXTRAER. Cuando la información haya sido extraída aparecerá el mensaje: "INFORMACIÓN EXTRAÍDA"

3.4 PROCESOS DE BORRADO.

El proceso de borrado consiste en borrar de forma parcial o total a los usuarios que están almacenados en el sistema, desde esta opción también se podrán borrar los registros creados en la base de datos.

1.- Para ingresar a esta opción digite el botón ADMINISTRADOR, seguidamente ingrese la Clave de Administrador (10031723).

2.- Se desplegará en el menú las siguientes opciones: Enrolar Usuario, Extraer Información y Opciones de Borrado. Se elegirá la opción OPCIONES DE BORRADO

3.- Se desplegará una pantalla como la que indica la figura 3.9 con estas opciones: Borrar U, Borrar T, Borrar Reg.

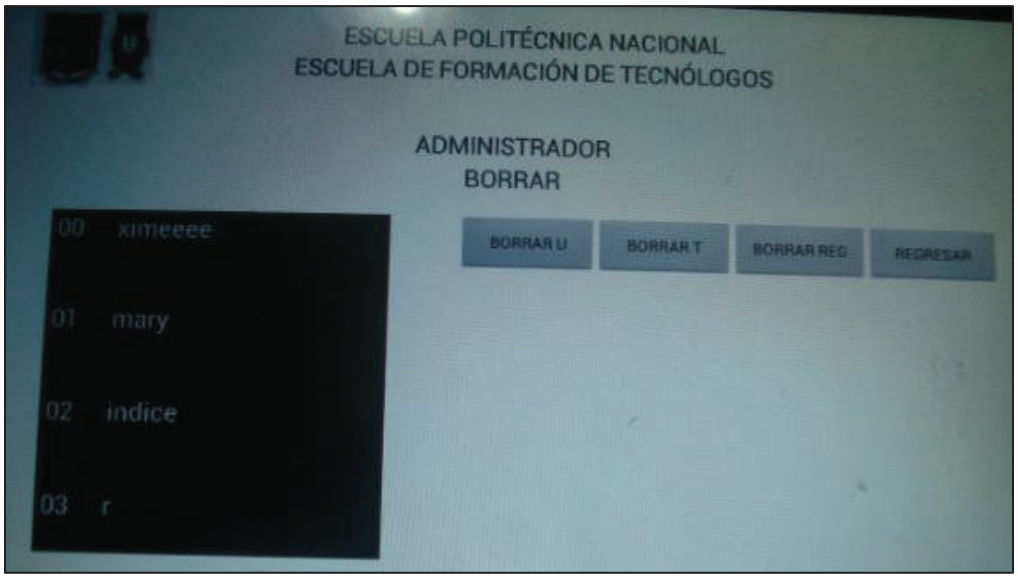


Figura 3.9: Opciones de borrado

4.- Con la opción BORRAR U, se podrá borrar únicamente un usuario para lo cual se deberá escoger en la pantalla de color negro en donde se encuentra la lista de individuos registrados como lo muestra la figura 3.10. El sistema mostrará el mensaje de confirmación a esta

petición: “BORRAR A: xxxx”. Si desea continuar con el proceso presione BORRAR U; nuevamente el sistema pedirá que se confirme su solicitud mostrando el mensaje: “CONFIRMA BORRAR A: xxxx” si su respuesta es positiva presione nuevamente BORRAR U; caso contrario presione REGRESAR.

Para finalizar este proceso el sistema mostrará el mensaje: “BORRADO FINALIZADO”

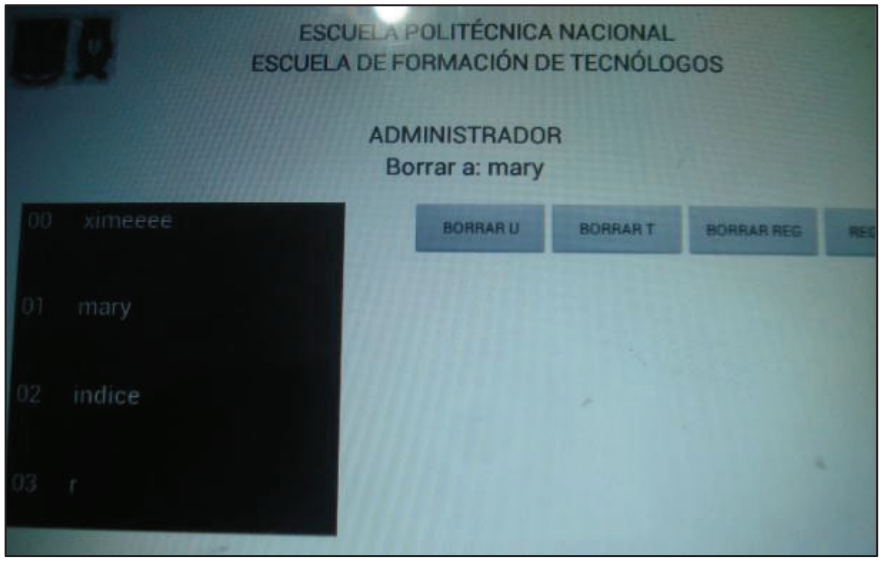


Figura 3.10: Selección del usuario a ser borrado

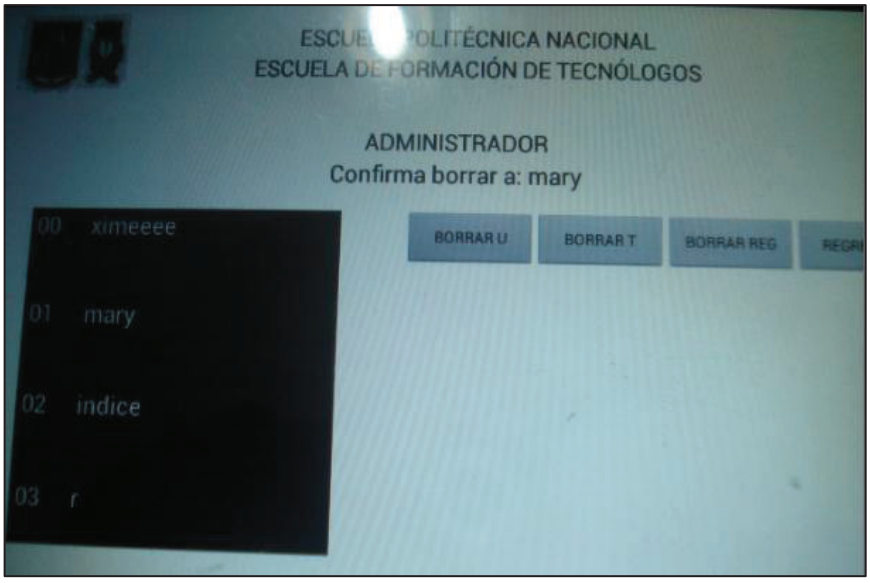


Figura 3.11: Confirmación de borrado de un usuario

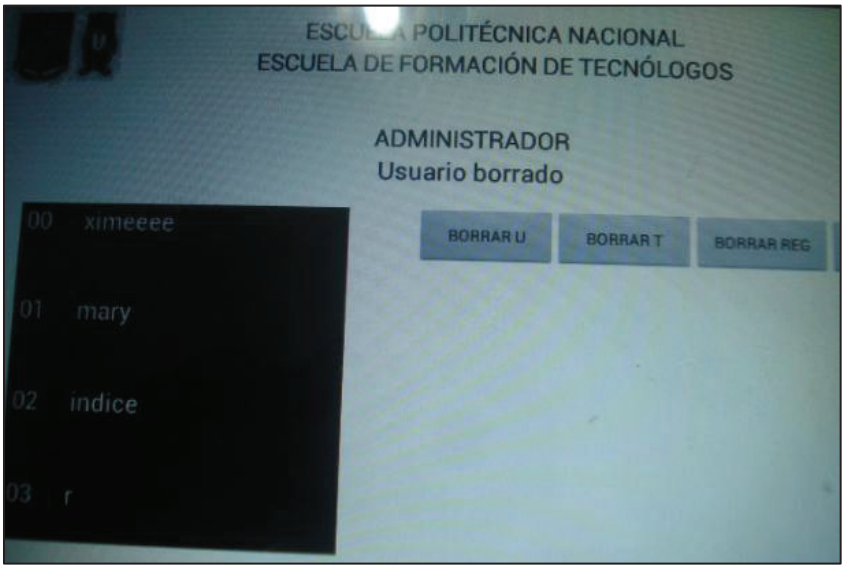


Figura 3.12: Usuario borrado

Nota: La verificación de que un usuario ha sido borrado se la podrá realizar digitando la opción REGRESAR e ingresando nuevamente a las OPCIONES DE BORRADO.

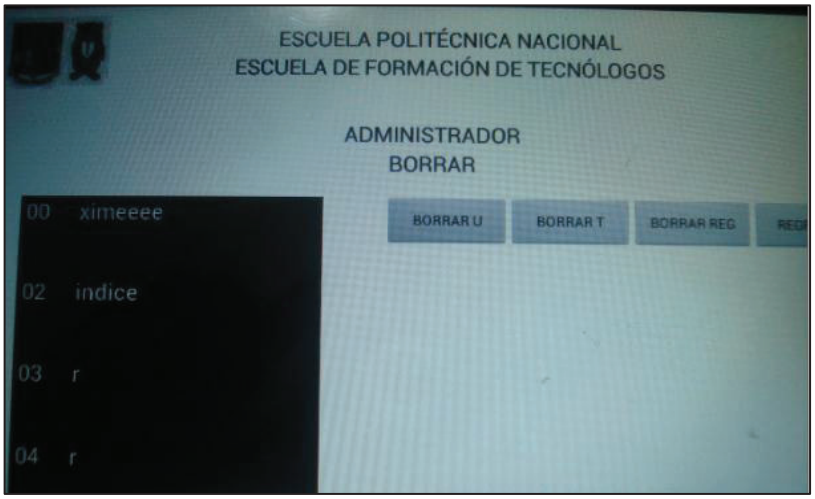


Figura 3.13: Pantalla de verificación de borrado de un usuario

5.- Con la opción BORRAR T, se podrá realizar el borrado de todos los usuarios almacenados en la base de datos del sistema.

Se deberá digitar el botón BORRAR T, a continuación el sistema mostrará un mensaje de confirmación: “DESEA BORRAR TODOS LOS USUARIOS”, si desea continuar con este proceso deberá presionar nuevamente BORRAR T, una vez más el sistema pedirá la confirmación de su petición con el mensaje: “CONFIRMA BORRAR TODOS LOS USUARIOS” si es así presione de nuevo BORRAR T, caso contrario presione REGRESAR.

Para finalizar este proceso se mostrará en la pantalla el siguiente mensaje: “TODOS LOS USUARIOS HAN SIDO BARRADOS”.

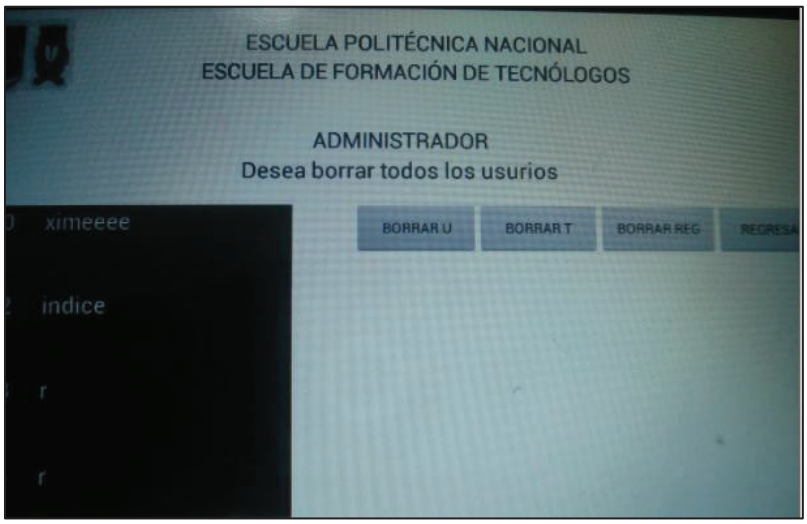


Figura 3.14: Botón BORRAR T

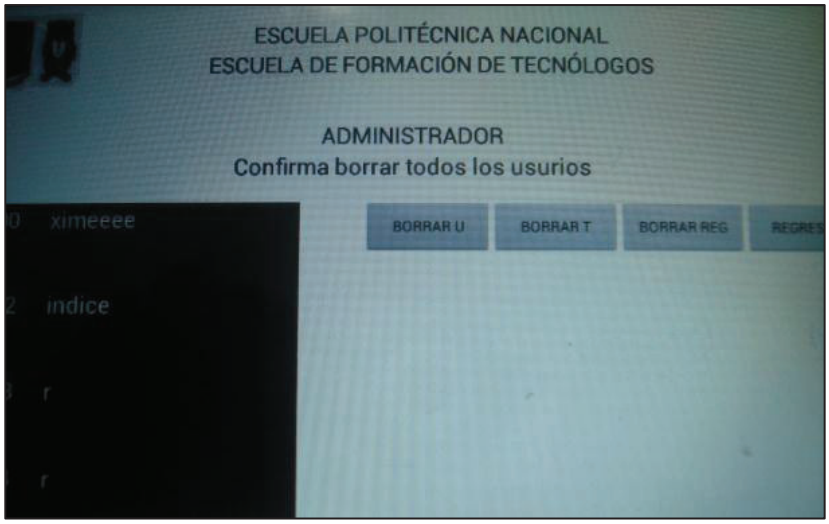


Figura 3.15: Confirmación de borrado de todos los usuarios

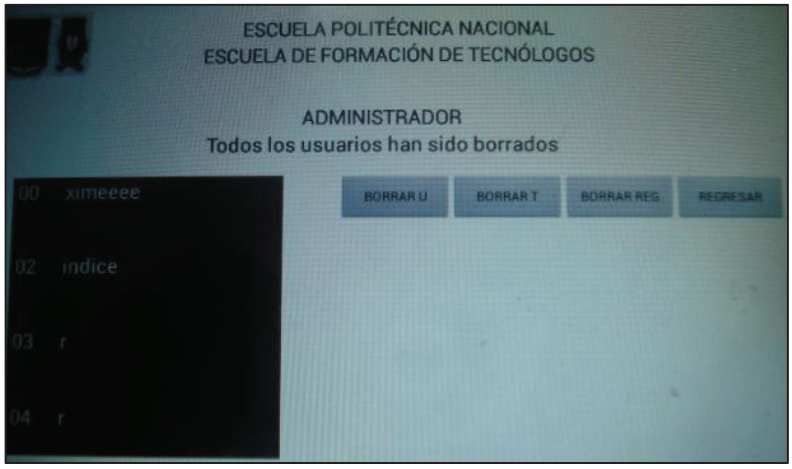


Figura 3.16: Mensaje de usuarios borrados

Nota: La verificación de que todos los usuarios han sido borrados se la podrá realizar digitando la opción REGRESAR e ingresando nuevamente a las OPCIONES DE BORRADO.

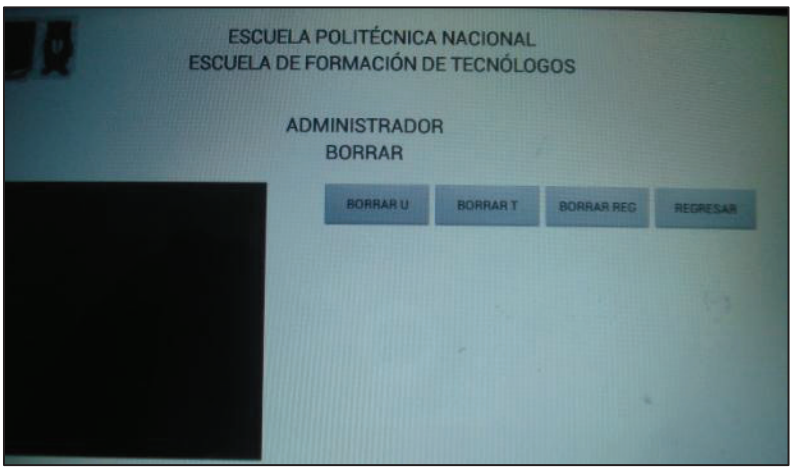


Figura 3.17: Verificación de borrado de todos los usuarios

6.- Con la opción BORRAR REG, el administrador podrá borrar los registros (Nombre del docente, Tema dictado, Fecha y Hora de registro) existentes en la base de datos.

Para realizar este proceso se debe digitar el botón BORRAR REG, a continuación se mostrará un mensaje de confirmación: “DESEA BORRAR TODOS LOS REGISTROS” si desea

continuar con esta actividad presione nuevamente BORRAR REG; de igual forma se verificará su solicitud con el mensaje: “CONFIRMA BORRAR TODOS LOS REGISTROS” si desea continuar presione por última vez BORRAR REG. Posteriormente se mostrará el mensaje: “TODOS LOS REGISTROS HAN SIDO BORRADOS” lo cual indica la finalización del proceso.

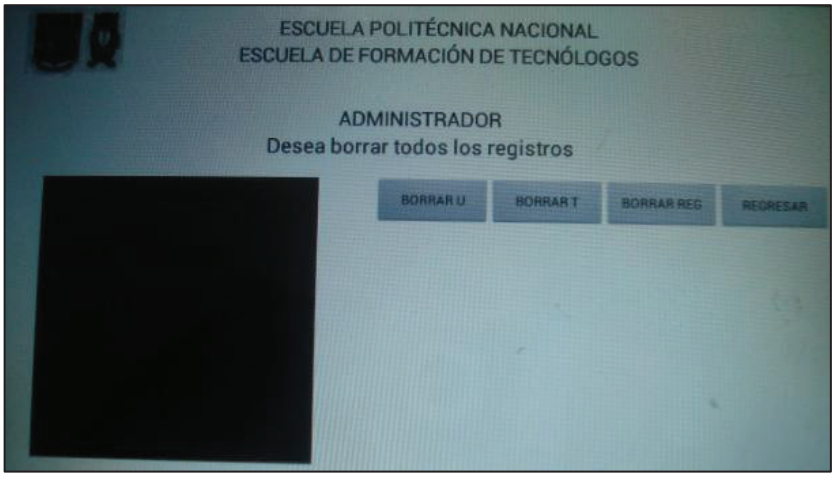


Figura 3.18: Botón BORRAR REG

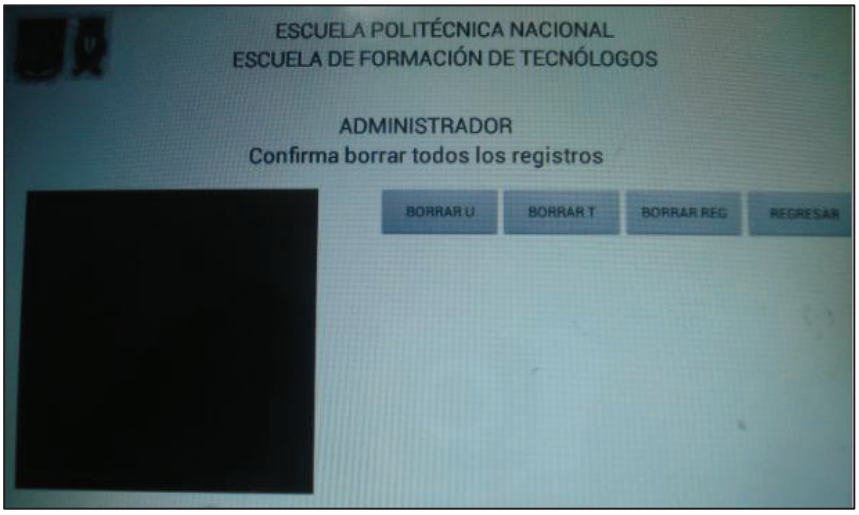


Figura 3.19: Confirmación de borrado de registros

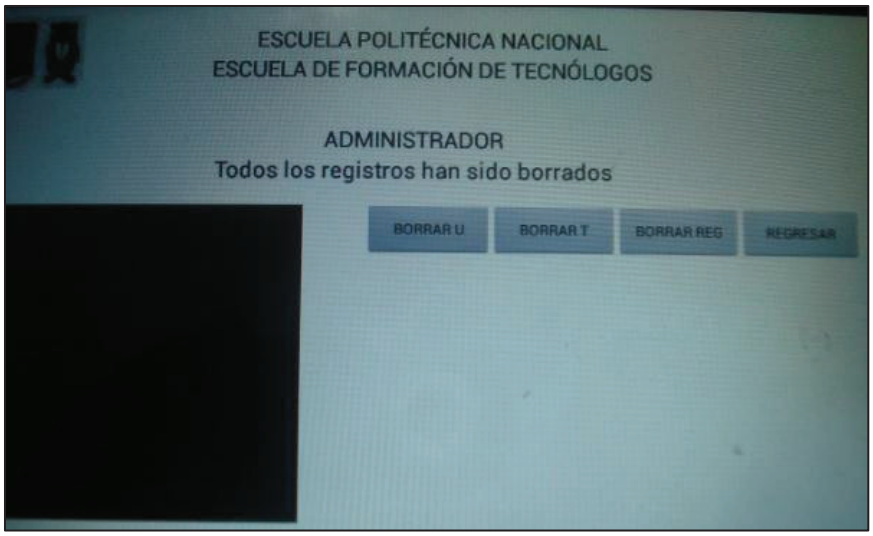


Figura 3.20: Mensaje de borrado de los registros

3.5 PROCESO DE IDENTIFICACIÓN DE USUARIOS.

Con la opción de IDENTIFICACIÓN, el usuario previamente enrolado podrá ingresar al sistema y generar la base de datos en la que consta: Nombre del docente, Tema dictado, Fecha y hora de registro. Para realizar el proceso de identificación se procederá con los siguientes pasos:

1.- En la pantalla principal se escogerá la opción IDENTIFICACIÓN

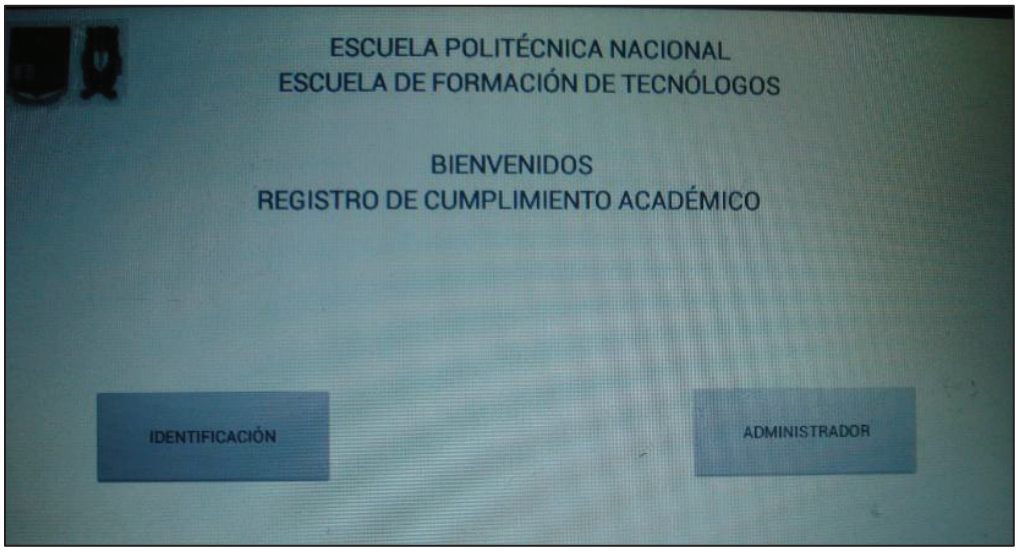


Figura 3.21: Pantalla de Inicio

2.- A continuación el sistema mostrará el siguiente mensaje "COLOQUE SU DEDO ÍNDICE" inmediatamente el sensor biométrico empezará a emitir 10 parpadeos consecutivos dentro de los cuales el usuario deberá colocar su dedo en el lector de huella digital.

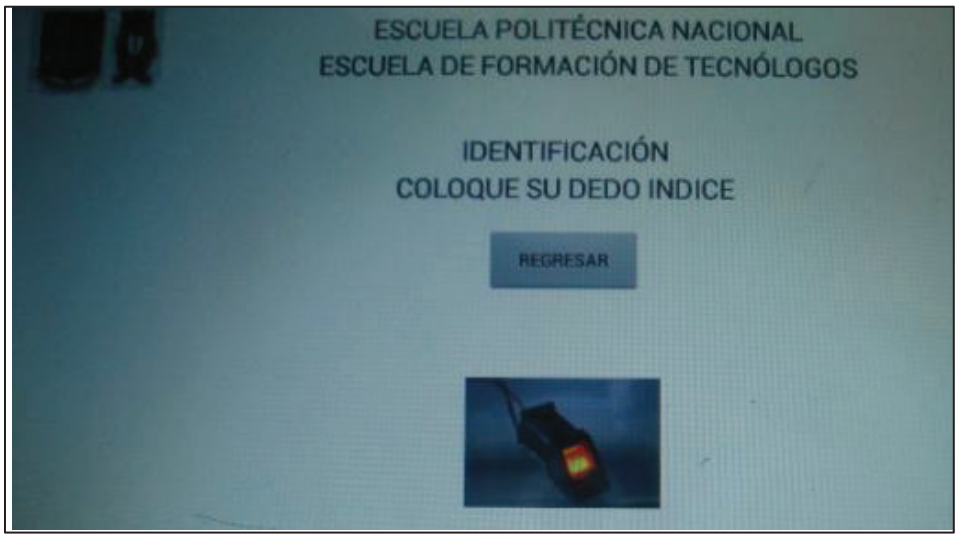


Figura 3.22: Mensaje de colocación del dedo en el Sensor Biométrico

Nota: Si el tiempo se agota el sistema emitirá un mensaje de: "TIEMPO AGOTADO" y volverá a la pantalla de inicio.

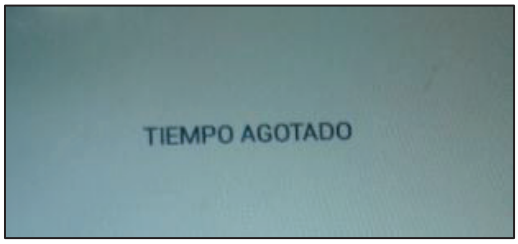


Figura 3.23: Mensaje de Tiempo Agotado

3.- Una vez que el sensor biométrico haya identificado y autenticado al usuario se mostrará la pantalla de la figura 3.24. En donde el docente deberá ingresar el Tema de clase a dictar.

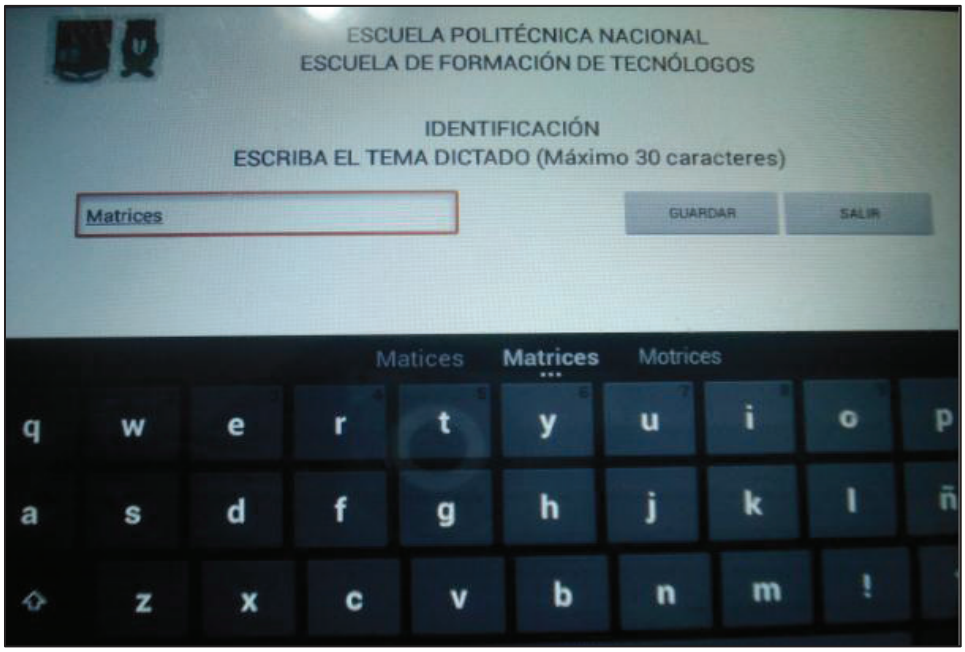


Figura 3.24: Ingresar Tema de clase

Nota: Se deberá ingresar un máximo de 30 caracteres.

4.- Posteriormente se digitará el botón de GUARDAR.

5.- Al finalizar el proceso el sistema emitirá el siguiente mensaje: “REGISTRO EXITOSO”.

CAPÍTULO 4.

CONCLUSIONES, RECOMENDACIONES Y ANEXOS

4.1 CONCLUSIONES

En este Sistema de Registro Académico se ha utilizado un sensor de huellas dactilares como medio de identificación, un STM32F4 para procesar la información, y una Tablet como interfaz gráfica del usuario. El sistema apuesta para su implementación en instituciones públicas y privadas como un dispositivo de registro de cumplimiento académico o laboral. Las principales conclusiones se presentan a continuación:

- ✓ La primera de ellas se refiere a la comunicación serial. La cual mediante pruebas llevadas a cabo ha demostrado ser una técnica rápida y eficaz mediante la cual dos dispositivos electrónicos totalmente independientes comparten información desde comandos de control hasta memoria de datos.

- ✓ Se puede programar una aplicación en un celular o Tablet con sistema operativo Android para utilizar su pantalla táctil con el fin de servir como interfaz gráfica a cualquier otro dispositivo con el que se esté comunicando.

- ✓ Una característica y probablemente la más ventajosa del STM32F4 Discovery es que es un dispositivo electrónico completamente independiente. No necesita circuitería externa para su funcionamiento ni para compilar un programa en él.

- ✓ Un dispositivo de reconocimiento biométrico enfocado a huellas dactilares es el método más eficaz existente para identificación. Tiene la ventaja frente a otros dispositivos debido a su eficiencia, versatilidad y precio accesible.

4.2 RECOMENDACIONES

- ✓ Familiarizarse con la utilización de hardware y software necesarios para probar los puertos destinados para comunicación serial de un dispositivo que cuente con dichos puertos.

- ✓ Diseñar aplicaciones sencillas en el entorno de desarrollo Android Studio ya que es un software libre que permite programar y ejecutar instrucciones que pueden ser visualizadas a través de la pantalla táctil de un dispositivo Android.

- ✓ Grabar y probar programas en el STM32F4 que se encuentran disponibles en la web ya que éstos contienen librerías muy útiles que servirán para desarrollar prácticamente cualquier proyecto.

- ✓ La información que se descarga del sistema es de tipo texto, en dicha información se refleja los datos que se han registrado en él. Esto se llevará a cabo mediante una memoria externa por lo cual se debe tener sumo cuidado con el extravío o mal uso de la misma.

- ✓ Se recomienda al usuario utilizar el dedo índice para el enrolamiento e identificación, de esta manera es menos probable una confusión u olvido al momento del registro académico.

- ✓ Se debe colocar el dedo índice en el centro del sensor biométrico para asegurar la correcta autenticación e identificación para lograr un registro rápido y eficaz.

REFERENCIAS BIBLIOGRÁFICAS

Número	Tema
1	"Biometría" • Marino Tapiador Mateos, Juan A. Sigüenza Pizarro. "Tecnologías Biométricas Aplicadas a la Seguridad", Primera Edición, Alfa Omega Grupo Editor, Septiembre 2005. • www.biometria.gov.ar
2	"Tipos de Sistemas Biométricos" • http://redyseguridad.fip.unam.mx/proyectos/biometria/clasificacionsistemas
3	"Sensor de Huellas Dactilares" • Escuela Politécnica Nacional, ESFOT, Electrónica y Telecomunicaciones, Jorge Bayas -Luis Molina. • https://es.wikipedia.org/wiki/Sensor_de_huella_digital
4	"Microcontrolador" • David A. Patterson, John L. Hennessy,"Estructura y Diseño de Computadoras", Cuarta Edición Original, Editorial Reverté, Barcelona 2011.
5	"Tablet" • https://es.wikipedia.org/wiki/Tableta_(computadora)
6	"Comunicación Serial" • http://digital.ni.com/public.nsf/allkb/ • https://galaxi0.wordpress.com/el-puerto-serial/

7	"STM32F4 Discovery Review" • Ricardo Cayssials, "Sistemas Embebidos en FPGA", Primera Edición, Alfa Omega Grupo Editor Argentino, 2014. • Especificaciones Técnicas STM32F4 Discovery Review
8	"Sensor Biométrico ZFM20" • Especificaciones Técnicas Sensor Biométrico ZFM20
9	"Circuito Auxiliar" • Ferran Reverter, Ramón Pallás Areny. "Circuitos de Interfaz Directa Sensor –Microprocesador", Primera Edición, Alfa Omega Grupo Editor, México Enero 2009.
10	**"**Herramientas de Programación**"** • http://www.coocox.org/ • https://es.wikipedia.org/wiki/Android_Studio
11	**"**Programación del Microcontrolador" • http://mikrocontroller.bplaced.net/wordpress/?page_id=744
12	**"**Sentencias utilizadas en la Programación**"** • www.monografias.com • Escuela Politécnica Nacional, ESFOT, Electrónica y Telecomunicaciones, Galo Chavarrea – Ana Chiluisa.

4.3 ANEXOS

ANEXO A. PROGRAMA DEL STM32F4

```
#include "main.h"
#include "stm32_ub_led.h"
#include "stm32_ub_button.h"
#include "stm32_ub_usb_msc_host.h"
#include "stm32_ub_uart.h"
#include "stm32_ub_string.h"
#include "stm32_av_zfm20x.h"

// Globale Variable
FIL myFile;                              // permite manejar los archivos tipo txt.
char buffSD[50];
char nameSD[15];
char userName[30];
uint16_t buffRead;
int auxTodos;
int noRep = 0;
int usbP = 0;

void delete_Reg(void);
int ext_Reg(void);
int ext_Inf(void);
void write_to_USB(void);
void write_to_SD(void);
uint16_t exist_US(void);    //pregunta la dirección de memoria disponible en el
biométrico para enrolar un usuario

int delete_SD(uint16_t DirFile);            // barrar datos de la SD
int read_User_SD(uint16_t nUser);
int read_User_Name_SD(uint16_t nUser);
void write_Name_SD(void);

uint8_t write_ok = 0;
UART_RXSTATUS_t check;
char buff[50];

SystemInit();                               // Inicialización del sistema
UB_Uart_Init();                             // inicialización del UART
AV_Zfm20x_Init();                           // inicialización del biométrico


UB_Led_Init();                              // inicialización LEDs
UB_Button_Init();                           //inicialización  Button
UB_Fatfs_Init();                            // inicialización  FATFS-System

//manejo del micro SD
UB_USB_MSC_HOST_Init();         //inicialización del puerto USB OTG como MSC-HOST

//leer / escribir en una unidad flash USB
int bus = 0;
int ident = 0;
int var = 0;
int16_t dato = 0;
```

```
while (1) {

check = UB_Uart_ReceiveString(COM3, buff); //chequea si se recibió algo por la
tablet (COM3)
if (check == RX_READY) {
ident = buff[0];// lo que envía la tablet empieza con una letra y le sigue el dato
for (var = 0; var < 49; ++var) {
buff[var] = buff[var + 1];           //nos entrega el dato desde la posición uno
}
dato = UB_String_DezStringToInt(buff);


//PARA ENROLAR
if (ident == 'E') {
dato = exist_US();//
sprintf(nameSD, "0:/%d.txt", dato); // crea el nombre (dirección de memoria
disponible) q se le va asignar al archivo

//tipo txt del usuario en la SD
//sprintf Crea una cadena de texto a partir de varios datos, siguiendo un cierto
código de formato.
while (1) {
check = UB_Uart_ReceiveString(COM3, buff);
if (check == RX_READY) {
UB_Led_On(LED_ORANGE);
ident = buff[0];
for (var = 0; var < 49; ++var) {
buffSD[var] = buff[var + 1];//pone en buff SD los datos del archivo
}
break;
}
}
write_to_SD(); //se guarda el primer parámetro (nombre) del archivo en la SD min
19.10

while (1) {
check = UB_Uart_ReceiveString(COM3, buff);
if (check == RX_READY) {
UB_Led_On(LED_ORANGE);
ident = buff[0];
for (var = 0; var < 49; ++var) {
buffSD[var] = buff[var + 1];
}
break;
}
}
write_to_SD();

while (1) {
check = UB_Uart_ReceiveString(COM3, buff);
if (check == RX_READY) {
UB_Led_On(LED_ORANGE);
ident = buff[0];
for (var = 0; var < 49; ++var) {
```

```
buffSD[var] = buff[var + 1];
}
break;
}
}
write_to_SD();

while (1) {
check = UB_Uart_ReceiveString(COM3, buff);
if (check == RX_READY) {
UB_Led_On(LED_ORANGE);
ident = buff[0];
for (var = 0; var < 49; ++var) {
buffSD[var] = buff[var + 1];
}
break;
}
}
write_to_SD();

while (1) {
check = UB_Uart_ReceiveString(COM3, buff);
if (check == RX_READY) {
UB_Led_On(LED_ORANGE);
ident = buff[0];
for (var = 0; var < 49; ++var) {
buffSD[var] = buff[var + 1];
}
break;
}
}
write_to_SD();

while (1) {
check = UB_Uart_ReceiveString(COM3, buff);
if (check == RX_READY) {
UB_Led_On(LED_ORANGE);
ident = buff[0];
for (var = 0; var < 49; ++var) {
buffSD[var] = buff[var + 1];
}
break;
}
}
write_to_SD();

while (1) {
check = UB_Uart_ReceiveString(COM3, buff);
if (check == RX_READY) {
UB_Led_On(LED_ORANGE);
ident = buff[0];
for (var = 0; var < 49; ++var) {
buffSD[var] = buff[var + 1];
}
break;
```

```
}
}
write_to_SD();

//función enrolar usuario "biométrico"
AV_Zfm20x_Delay(0xFFFFFF);
if (AV_Zfm20x_Enrolar(dato) == 1) {
AV_Zfm20x_Delay(0xFFFFF);
UB_Uart_SendString(COM3, "e1", LFCR);//envía a la tablet enrolado exitoso
} else {
delete_SD(dato);
AV_Zfm20x_Delay(0xFFFFF);
UB_Uart_SendString(COM3, "e0", LFCR);//envía a la tablet enrolado no exitoso
}
}

//PARA BUSCAR (identificación)
if (ident == 'B') {
bus = AV_Zfm20x_Buscar();//obtenemos el número del usuario y lo ponemos en bus
if (bus == -1) {
UB_Uart_SendString(COM3, "b0", LFCR);                    //tiempo agotado
} else if (bus == -2) {
UB_Uart_SendString(COM3, "b2", LFCR);                    //usuario no adimitido
} else {
UB_Uart_SendString(COM3, "b1", LFCR);                   // usuario admitido

while (1) {
check = UB_Uart_ReceiveString(COM3, buffSD);
if (check == RX_READY) {
ident = buffSD[0];
for (var = 0; var < 49; ++var) {
buffSD[var] = buffSD[var + 1];
}
if (buffSD[0] == 'm') { //si lo que se recibe es el tema dictado (dato con
cabecera m)
UB_Led_On(LED_RED);
break;

//para crear el nombre del archivo registro (formato fecha)
} else {
UB_Led_On(LED_ORANGE);
nameSD[0] = '0';
nameSD[1] = ':';
nameSD[2] = '/';
nameSD[11] = '.';
nameSD[12] = 't';
nameSD[13] = 'x';
nameSD[14] = 't';

//se escoge la parte del buffer q corresponde a la fecha
for (var = 3; var < 11; ++var) {

if (var < 9) {
nameSD[var] = buffSD[var - 3];
} else {
```

```
nameSD[var] = buffSD[var - 3 + 2];
}
}
UB_Uart_SendString(COM3, nameSD, LFCR);
write_to_SD();   //leemos en nombre del usuario según el número obtenido en Buscar
read_User_Name_SD(bus);
write_Name_SD();
break;
}
}
}
}
}

//PARA BORRAR
if (ident == 'U') {
//Barrido de usuarios en SD
if (noRep == 0) {
uint16_t var = 0;
for (var = 0; var < 100; ++var) {
if (read_User_SD(var) == 1) {                //si existe el usuario envía 1
UB_Uart_SendString(COM3, userName, LFCR);    //envía la lista de usuarios
AV_Zfm20x_Delay(0xFFFFF);
}
}
noRep = 1;
}

while (1) {
check = UB_Uart_ReceiveString(COM3, buff);
if (check == RX_READY) {
ident = buff[0];
if (ident == 'U') {
for (var = 0; var < 49; ++var) {
buff[var] = buff[var + 1];
}
dato = UB_String_DezStringToInt(buff);

AV_Zfm20x_BorrarD(dato);
delete_SD(dato);
}
if (ident == 'T') {
AV_Zfm20x_BorrarT();
uint16_t var = 0;
for (var = 0; var < 100; ++var) {
delete_SD(var);
}
}

if (ident == 'R') {
delete_Reg();
}

if (ident == 'S') {
noRep = 0;                                   //regresamos al Barrido
```

```
break;
}

}
}
}

//PARA EXTRAER INFORMACÓN
if (ident == 'I') {
if (ext_Inf() == 1) {
ext_Reg();
UB_Uart_SendString(COM3, "i1", LFCR);     //información extraída
} else {
UB_Uart_SendString(COM3, "i0", LFCR);    //información no extraída
}
}
}

//chekea si la USB esta insertada o no
if (UB_USB_MSC_HOST_Do() == USB_MSC_DEV_CONNECTED) {    //si USB conectada
usbP = 1;
UB_Led_On(LED_GREEN);
UB_Led_Off(LED_RED);
} else {
usbP = 0;
UB_Led_On(LED_RED);
UB_Led_Off(LED_GREEN);
}
}
}

void delete_Reg(void) {
uint16_t anio = 0;
uint16_t mes = 0;
uint16_t dia = 0;
char nameSDD[20];

for (anio = 15; anio < 17; ++anio) {
for (mes = 1; mes < 13; ++mes) {
for (dia = 1; dia < 32; ++dia) {

if ((dia < 10) & (mes < 10)) {
sprintf(nameSDD, "0:/0%d-0%d-%d.txt", dia, mes, anio);
} else if ((dia < 10) & (mes >= 10)) {
sprintf(nameSDD, "0:/0%d-%d-%d.txt", dia, mes, anio);
} else if ((dia >= 10) & (mes < 10)) {
sprintf(nameSDD, "0:/%d-0%d-%d.txt", dia, mes, anio);
} else {
sprintf(nameSDD, "0:/%d-%d-%d.txt", dia, mes, anio);
}

if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
if (UB_Fatfs_DelFile(nameSDD) == FATFS_OK) {
}
}
```

```
}
}
}
}

int ext_Reg(void) {
char bufAux[512];
uint16_t anio = 0;
uint16_t mes = 0;
uint16_t dia = 0;
uint16_t size = 0;
uint16_t size2 = 0;
uint16_t read = 0;
char nameSDD[20];
int aseg = 0;
int cR = 1;
int ccr = 0;
int c = 1;
if (1 == 1) {
UB_Led_On(LED_BLUE);
if (UB_Fatfs_Mount(USB_1) == FATFS_OK) {
if (UB_Fatfs_OpenFile(&myFile, "1:/REGISTRO.txt", F_WR_CLEAR)
== FATFS_OK) {

UB_Fatfs_WriteString(&myFile, "Registro de actividades: ");
UB_Fatfs_WriteString(&myFile, "   ");
UB_Fatfs_CloseFile(&myFile);
}
}

for (anio = 15; anio < 17; ++anio) {
for (mes = 1; mes < 13; ++mes) {
for (dia = 1; dia < 32; ++dia) {

if ((dia < 10) & (mes < 10)) {
sprintf(nameSDD, "0:/0%d-0%d-%d.txt", dia, mes, anio);
} else if ((dia < 10) & (mes >= 10)) {
sprintf(nameSDD, "0:/0%d-%d-%d.txt", dia, mes, anio);
} else if ((dia >= 10) & (mes < 10)) {
sprintf(nameSDD, "0:/%d-0%d-%d.txt", dia, mes, anio);
} else {
sprintf(nameSDD, "0:/%d-%d-%d.txt", dia, mes, anio);
}

if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
if (UB_Fatfs_OpenFile(&myFile, nameSDD, F_RD)   //crea el archivo solo de lectura
== FATFS_OK) {
size = UB_Fatfs_FileSize(&myFile);             //lee la dimensión del archivo
UB_Fatfs_CloseFile(&myFile);
for (cR = 1; size > 0; ++cR) {
if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
if (UB_Fatfs_OpenFile(&myFile, nameSDD,
F_RD) == FATFS_OK) {
aseg = 1;                                      //si se pudo leer
if (size <= 510) {
```

```
cR = cR - 1;
}
for (ccr = 0; ccr < cR; ++ccr) {
UB_Fatfs_ReadBlock(&myFile, bufAux,
510, read);//abre archivo bufAux
}
if (size <= 510) {
UB_Fatfs_ReadBlock(&myFile, bufAux,
size, read);
size2 = size;
size = 0;
} else {
size = size - 510;
}
UB_Fatfs_CloseFile(&myFile);
}
}
if (aseg == 1) {
if (UB_Fatfs_Mount(USB_1) == FATFS_OK) {
if (UB_Fatfs_OpenFile(&myFile,
"1:/REGISTRO.txt", F_WR)
== FATFS_OK) {                                  //abre el archivo REGISTRO (sobre
escribe)
if (size > 0) {
UB_Fatfs_WriteBlock(&myFile,
bufAux, 510, read);

} else {
UB_Fatfs_WriteBlock(&myFile,
bufAux, size2, read);                    //escribe el archivo en la USB
}
UB_Fatfs_CloseFile(&myFile);
aseg = 0;
}
}
}
}
}
}
}
}
}

UB_Fatfs_UnMount(USB_1);                //desmonta la USB
UB_Fatfs_UnMount(MMC_0);
UB_Led_Off(LED_BLUE);
return 1;

} else {
UB_Led_On(LED_GREEN);
return -1;
}
}

int ext_Inf(void) {
```

```
char bufAux[512];
uint16_t varr = 0;
uint16_t size = 0;
uint16_t read = 0;
char nameSDD[20];
int aseg = 0;

if (usbP == 1) {                                                    //si la USB está
conectada
UB_Led_On(LED_BLUE);
if (UB_Fatfs_Mount(USB_1) == FATFS_OK) {
if (UB_Fatfs_OpenFile(&myFile, "1:/USUARIOS.txt", F_WR_CLEAR)   //crea el archivo
== FATFS_OK) {

UB_Fatfs_WriteString(&myFile, "Lista de usuarios: ");
UB_Fatfs_WriteString(&myFile, "   ");
UB_Fatfs_CloseFile(&myFile);
}
}

for (varr = 0; varr < 100; ++varr) {
sprintf(nameSDD, "0:/%d.txt", varr);                            //formamos el nombre
if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {//montamos la SD
if (UB_Fatfs_OpenFile(&myFile, nameSDD, F_RD) == FATFS_OK) {//abre archivo de
lectura
aseg = 1;
size = UB_Fatfs_FileSize(&myFile);            //obtiene la dimensión del archivo
UB_Fatfs_ReadBlock(&myFile, bufAux, size, read);//guarda el archivo en buffer Aux
UB_Fatfs_CloseFile(&myFile);
}
}
if (aseg == 1) {
if (UB_Fatfs_Mount(USB_1) == FATFS_OK) {
if (UB_Fatfs_OpenFile(&myFile, "1:/USUARIOS.txt", F_WR)
== FATFS_OK) {
UB_Fatfs_WriteBlock(&myFile, bufAux, size, read); //escribimos el archivo en la
USB
UB_Fatfs_WriteString(&myFile, "   ");
UB_Fatfs_CloseFile(&myFile);
aseg = 0;
}
}
}
}

UB_Fatfs_UnMount(USB_1);
UB_Fatfs_UnMount(MMC_0);
UB_Led_Off(LED_BLUE);
return 1;

} else {
UB_Led_On(LED_GREEN);
return -1;                                  //si la USB se desconecta
}
}
```

```
int read_User_SD(uint16_t nUser) {
char buffDir[50];
char buffAux[50];
uint16_t var = 0;

if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
sprintf(buffDir, "0:/%d.txt", nUser);
if (UB_Fatfs_OpenFile(&myFile, buffDir, F_RD) == FATFS_OK) {
UB_Fatfs_ReadString(&myFile, userName, 50);
if (nUser < 10) {
userName[0] = 0 + 48;
userName[1] = nUser + 48;
}

if ((nUser >= 10) && (nUser < 100)) {
userName[0] = nUser / 10 + 48;
userName[1] = nUser - (nUser / 10) * 10 + 48;
}

UB_Fatfs_CloseFile(&myFile);
return 1;

} else {
return -1;
}
}
}

int read_User_Name_SD(uint16_t nUser) {
char buffDir[50];
char buffAux[50];
uint16_t var = 0;

if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
sprintf(buffDir, "0:/%d.txt", nUser);
if (UB_Fatfs_OpenFile(&myFile, buffDir, F_RD) == FATFS_OK) {
UB_Fatfs_ReadString(&myFile, userName, 50);
UB_Fatfs_CloseFile(&myFile);
return 1;

} else {
return -1;
}
}
}

int delete_SD(uint16_t DirFile) {
char buffDir[20];
sprintf(buffDir, "0:/%d.txt", DirFile);
if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
if (UB_Fatfs_DelFile(buffDir) == FATFS_OK) {
return 1;
}
}
return -1;}
```

```
uint16_t exist_US(void) {
char buffDir[20];

uint16_t var = 0;
if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
for (var = 0; var < 255; ++var) {//barrido
sprintf(buffDir, "0:/%d.txt", var);
if (UB_Fatfs_OpenFile(&myFile, buffDir, F_RD) == FATFS_OK) { //abre el archivo si
a caso existe
UB_Fatfs_CloseFile(&myFile);
} else {
return var;      //si no existe envía el número (dirección)disponible para el
usuario
}
}
}
return -1;
}


void write_to_SD(void) {
// MMC-Media mounten
if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
UB_Led_On(LED_ORANGE);
// File zum schreiben im root neu anlegen

//if (UB_Fatfs_OpenFile(&myFile, "0:/SD_Fkkit.txt", F_WR_CLEAR)
if (UB_Fatfs_OpenFile(&myFile, nameSD, F_WR_NEW) == FATFS_OK) {
// ein paar Textzeilen in das File schreiben
UB_Fatfs_WriteString(&myFile, buffSD);

// File schliessen
UB_Fatfs_CloseFile(&myFile);
}
// Media unmounten
UB_Fatfs_UnMount(MMC_0);
UB_Led_Off(LED_ORANGE);
}
}


void write_Name_SD(void) {
// MMC-Media mounten
if (UB_Fatfs_Mount(MMC_0) == FATFS_OK) {
UB_Led_On(LED_ORANGE);
// File zum schreiben im root neu anlegen

//if (UB_Fatfs_OpenFile(&myFile, "0:/SD_Fkkit.txt", F_WR_CLEAR)
if (UB_Fatfs_OpenFile(&myFile, nameSD, F_WR_NEW) == FATFS_OK) {
// ein paar Textzeilen in das File schreiben
UB_Fatfs_WriteString(&myFile, userName);

// File schliessen
UB_Fatfs_CloseFile(&myFile);}
```

```
// Media unmounten
UB_Fatfs_UnMount(MMC_0);
UB_Led_Off(LED_ORANGE);
}
}

void write_to_USB(void) {
// USB-Media mounten
if (UB_Fatfs_Mount(USB_1) == FATFS_OK) {
UB_Led_On(LED_RED);
// File zum schreiben im root neu anlegen
if (UB_Fatfs_OpenFile(&myFile, "1:/USB_Fkkile.txt", F_WR_CLEAR)
== FATFS_OK) {
// ein paar Textzeilen in das File schreiben
UB_Fatfs_WriteString(&myFile, "USB-Textfile");
UB_Fatfs_WriteString(&myFile, "hier Zeile zwei");
UB_Fatfs_WriteString(&myFile, "ENDE");
// File schliessen
UB_Fatfs_CloseFile(&myFile);
}
// Media unmounten
UB_Fatfs_UnMount(USB_1);
UB_Led_Off(LED_RED);
}
}
```

ANEXO B. PROGRAMA DE LA TABLET

```
*****************SerialConsoleActivity**************

//LIBRERIAS UTILIZADAS EN TODAS LAS CLASES

import java.util.ArrayList;
import java.util.List;
import com.hoho.android.usbserial.driver.UsbSerialDriver;
import com.hoho.android.usbserial.driver.UsbSerialProber;
import com.hoho.android.usbserial.util.HexDump;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.os.SystemClock;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.Context;
import android.hardware.usb.UsbDevice;
import android.hardware.usb.UsbManager;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.TwoLineListItem;


public class SerialConsoleActivity extends Activity {
private final String TAG = SerialConsoleActivity.class.getSimpleName();
private static UsbSerialDriver sDriver = null; //para CREAR el driver del USB
serial
Button bIdent, bAdmin;                        //Declaración de botones
String mmm="";

//PARA ACITIVAR EL DRIVER USB SERIAL (PRESENTE EN TODAS LAS CLASES)

private final ExecutorService mExecutor = Executors.newSingleThreadExecutor();
private SerialInputOutputManager mSerialIoManager;
private final SerialInputOutputManager.Listener mListener =
new SerialInputOutputManager.Listener() {

@Override
public void onRunError(Exception e) {
```

```
Log.d(TAG, "Runner stopped.");
}

@Override
public void onNewData(final byte[] data) {
SerialConsoleActivity.this.runOnUiThread(new Runnable() {
@Override
public void run() {
SerialConsoleActivity.this.updateReceivedData(data);
}
});
}
};

//DONDE SE HACE LA PROGRAMACIÓN
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.serial_console);

bIdent = (Button) findViewById(R.id.IDENT);      //instanciamos los botones
bAdmin = (Button) findViewById(R.id.ADMIN);

bIdent.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {                  //escucha si el botón es presionado

showConsoleActivity(1,"BIENVENIDO USUARIO");   //muestra el mensaje de transición
}
});

bAdmin.setOnClickListener(new OnClickListener(){ //si el botón Admin es presionado
@Override
public void onClick(View v) {
showConsoleActivity(3,"BIENVENIDO ADMINISTRADOR"); //muestra el mensaje
}
});
}

//CUANDO SE SALE DE LA APLICACIÓN  (PRESENTE EN TODAS LAS CLASES)

@Override
protected void onPause() {
super.onPause();
stopIoManager();
if (sDriver != null) {
try {
sDriver.close();
} catch (IOException e) {
}
sDriver = null;
```

```
}
finish();
}

//PARA CUANDO SE ENTRA A LA APLICACIÓN (PRESENTE EN TODAS LAS CLASES)

@Override
protected void onResume() {
super.onResume();

Log.d(TAG, "Resumed, sDriver=" + sDriver);
if (sDriver == null) {
} else {
try {
sDriver.open();                          //ABRIMOS EL DRIVER

sDriver.setParameters(57600,8,UsbSerialDriver.STOPBITS_1,
UsbSerialDriver.PARITY_NONE);            //PARÁMETROS DEL PUERTO SERIAL
} catch (IOException e) {
Log.e(TAG, "Error setting up device: " + e.getMessage(), e);
try {
sDriver.close();
} catch (IOException e2) {
}
sDriver = null;
return;
}
}
onDeviceStateChange();
}

//GESTIÓN DE ENTRADAS Y SALIDAS (PRESENTE EN TODAS LAS CLASES)

private void stopIoManager() {
if (mSerialIoManager != null) {
Log.i(TAG, "Stopping io manager ..");
mSerialIoManager.stop();
mSerialIoManager = null;
}
}
//MANEJO DEL DRIVER USB
private void startIoManager() {
if (sDriver != null) {
Log.i(TAG, "Starting io manager ..");
mSerialIoManager = new SerialInputOutputManager(sDriver, mListener);
mExecutor.submit(mSerialIoManager);
}
}

private void onDeviceStateChange() {
stopIoManager();
```

```
startIoManager();
}

private void updateReceivedData(byte[] data) {   //donde se va a recibir los datos
mmm= new String(data);
}

//gestion de pantallas//
static void show(Context context, UsbSerialDriver driver) { //activa mostrar
pantallas
sDriver = driver;
final Intent intent = new Intent(context, SerialConsoleActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP                                    |
Intent.FLAG_ACTIVITY_NO_HISTORY);
context.startActivity(intent);
}

private void showConsoleActivity(int pag, String men) {   //activa navegar entre
pantallas
MainActivity.show(this, pag, men);
}
}
```

**********MainActivity**********

```
//navegación entre pantallas
private void showConsoleActivity(UsbSerialDriver driver) {

switch (pag){
case 0:
pag=0;
SerialConsoleActivity.show(this, driver);
break;
case 1:
pag=0;
Identificacion.show(this, driver);
break;
case 2:
pag=0;
TemaDictado.show(this, driver);
break;
case 3:
pag=0;
Administrador.show(this, driver);
break;
case 4:
pag=0;
AdministradorMenu.show(this, driver);
```

```
break;
case 5:
pag=0;
AdministradorEnrolar.show(this, driver);
break;
case 6:
pag=0;
AdministradorColocarDedo.show(this, driver);
break;
case 7:
pag=0;
AdministradorExtInfo.show(this, driver);
break;
case 8:
pag=0;
AdministradorBorrar.show(this, driver);
break;
default:
pag=0;
SerialConsoleActivity.show(this, driver);
break;
}
}

*****************Identificación*****************

Button bRegreso;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_identificacion);
bRegreso = (Button) findViewById(R.id.REG_IDENT);

bRegreso.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
showConsoleActivity(0,"IDENTIFICACIÓN FINALIZADA");
}
});
}


@Override
protected void onPause() {
super.onPause();
stopIoManager();
if (sDriver != null) {
try {
sDriver.close();
} catch (IOException e) {
```

```
// Ignore.
}
sDriver = null;
}
finish();
}

@Override
protected void onResume() {

super.onResume();
Log.d(TAG, "Resumed, sDriver=" + sDriver);
if (sDriver == null) {
} else {
try {
sDriver.open();
sDriver.setParameters(56700,8,UsbSerialDriver.STOPBITS_1,
UsbSerialDriver.PARITY_NONE);

} catch (IOException e) {
Log.e(TAG, "Error setting up device: " + e.getMessage(), e);
try {
sDriver.close();
} catch (IOException e2) {
// Ignore.
}
sDriver = null;
return;
}
}
onDeviceStateChange();

byte[] opt;
String se="B0\r";
opt = se.getBytes();
try {
sDriver.write(opt, 15);
} catch (IOException e) {

private void updateReceivedData(byte[] data) {

switch (data[1]){
case '0':
showConsoleActivity(0,"USUARIO NO ADMITIDO");
break;
case '1':
showConsoleActivity(2,"USUARIO ADMITIDO");
break;
case '2':
showConsoleActivity(0,"TIEMPO AGOTADO");
break;
```

```
default:
break;
}
}


********TemaDictado********

public class TemaDictado extends Activity {

Button bGuardarMateria, bSalirMateria;
EditText eTema;
SimpleDateFormat fecha = new SimpleDateFormat("dd-MM-yyyy");
SimpleDateFormat hora = new SimpleDateFormat("HH:mm:ss");


@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_tema_dictado);

bGuardarMateria = (Button) findViewById(R.id.GUARDAR_MD);
bSalirMateria = (Button) findViewById(R.id.SALIR_MD);
eTema = (EditText) findViewById(R.id.TEXTO_MD);

bGuardarMateria.setOnClickListener(new View.OnClickListener() {

String sFecha = fecha.format(new Date());
String sHora = hora.format(new Date());
@Override
public void onClick(View view) {
byte[] opt;

if(eTema.getText().toString().length()>29){
opt                    =                    ("M"+sFecha+"                    "+sHora+"
"+eTema.getText().toString().substring(0,28)+"\r").getBytes();

}else{
opt = ("M"+sFecha+" "+sHora+" "+eTema.getText().toString()+"\r").getBytes();
}

try {
sDriver.write(opt, 50);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
showConsoleActivity(0,"REGISTRO EXITOSO");
}
});
```

```
bSalirMateria.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
byte[] opt;
opt = ("Mm"+"\r").getBytes();

try {
sDriver.write(opt, 15);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
showConsoleActivity(0,"");



**********RegistroExitoso**********

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_registro_exitoso);
}


@Override
public boolean onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.menu_registro_exitoso, menu);
return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {

int id = item.getItemId();
if (id == R.id.action_settings) {
return true;
}

return super.onOptionsItemSelected(item);
}
}


**********Administrador**********

String clave="10031723";
EditText ePass;
Button bIngresarAdmin, bSalirAdmin;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_administrador);

ePass = (EditText) findViewById(R.id.PASS_AD);
bIngresarAdmin = (Button) findViewById(R.id.INGRESAR_AD);
bSalirAdmin = (Button) findViewById(R.id.SALIR_AD);

bIngresarAdmin.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
if(ePass.getText().toString().equals(clave)){
showConsoleActivity(4,"CLAVE CORRECTA");
}else{
showConsoleActivity(0,"CLAVE INCORRECTA");
}
}
});

bSalirAdmin.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
showConsoleActivity(0,"");
}
});
}
static void show(Context context, UsbSerialDriver driver) {
sDriver = driver;
final Intent intent = new Intent(context, Administrador.class);
intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP                                    |
Intent.FLAG_ACTIVITY_NO_HISTORY);
context.startActivity(intent);
}

private void showConsoleActivity(int pag, String men) {
MainActivity.show(this, pag, men);
}
}


**********AdministradorMenú**********

Button bEnrolar, bInfo, bSalirMenuAdmin, bBorrarAdmin;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_administrador_menu);

bEnrolar = (Button) findViewById(R.id.ENROLAR_AD);
```

```
bInfo = (Button) findViewById(R.id.EXT_INFO_AD);
bBorrarAdmin = (Button) findViewById(R.id.BORRAR_AD);
bSalirMenuAdmin = (Button) findViewById(R.id.SALIR_AD_MEN);

bEnrolar.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
showConsoleActivity(5,"ENROLAR USUARIO");
}
});

bInfo.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
showConsoleActivity(7,"EXTRAER INFORMACIÓN");
}
});

bBorrarAdmin.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
showConsoleActivity(8,"");
}
});

bSalirMenuAdmin.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
showConsoleActivity(0,"GRACIAS ADMINISTRADOR");
}
});    }
}

*********AdministradorBorrar******************

Button bBorrarT, bBorrarU, bRegresarB, bRegistro;
ListView lBorrar;
TextView tMensaje;
private ArrayAdapter<String> mBorrarData;
int confirmarU=0;
int confirmarT=0;
int confirmarR=0;
String mensaje="";
int usuario=0;
int noRep=0;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_administrador_borrar);

bBorrarU = (Button) findViewById(R.id.BORRAR_U);
```

```
bBorrarT = (Button) findViewById(R.id.BORRAR_T);
bRegresarB = (Button) findViewById(R.id.REGRESAR_B);
bRegistro = (Button) findViewById(R.id.B_REG);
lBorrar = (ListView) findViewById(R.id.listView);
mBorrarData = new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1);
lBorrar.setAdapter(mBorrarData);
tMensaje = (TextView) findViewById(R.id.textView4);

bBorrarU.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
//showConsoleActivity(4,"BORRADO FINALIZADO");
if((mensaje.length()<8)) {
}else{
switch (confirmarU){
case 0:
confirmarU=1;
tMensaje.setText("Confirma borrar a: "+mensaje.substring(8,mensaje.length()));
break;
case 1:
confirmarU=0;
usuario = Integer.parseInt(mensaje.substring(0,2));

byte[] opt;
String se="U"+ Integer.toString(usuario) +"\r";
opt = se.getBytes();
try {
sDriver.write(opt, 15);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

tMensaje.setText("Usuario borrado");
break;
default:
break;
}
}
}
});

bBorrarT.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
//showConsoleActivity(4,"BORRADO FINALIZADO");

switch (confirmarT){
case 0:
confirmarT=1;
tMensaje.setText("Desea borrar todos los usuarios");
```

```
break;
case 1:
confirmarT=2;
tMensaje.setText("Confirma borrar todos los usuarios");
break;
case 2:
confirmarT=0;

byte[] opt;
String se="T0\r";
opt = se.getBytes();
try {
sDriver.write(opt, 15);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

tMensaje.setText("Todos los usuarios han sido borrados");
break;
default:
break;
}
}
});

bRegistro.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
//showConsoleActivity(4,"BORRADO FINALIZADO");

switch (confirmarR){
case 0:
confirmarR=1;
tMensaje.setText("Desea borrar todos los registros");
break;
case 1:
confirmarR=2;
tMensaje.setText("Confirma borrar todos los registros");
break;
case 2:
confirmarR=0;

byte[] opt;
String se="R0\r";
opt = se.getBytes();
try {
sDriver.write(opt, 15);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
```

```
}
tMensaje.setText("Todos los registros han sido borrados");
break;
default:
break;
}
}
});

bRegresarB.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {

byte[] opt;
String se="S0\r";
opt = se.getBytes();
try {
sDriver.write(opt, 15);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
showConsoleActivity(4,"BORRADO FINALIZADO");
}
});

lBorrar.setOnItemClickListener(new ListView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
TextView textView = (TextView) view;

mensaje = textView.getText().toString();
tMensaje.setText("Borrar a: "+mensaje.substring(8,mensaje.length()));
}
});
}
onDeviceStateChange();

byte[] opt;
String se="U0\r";
opt = se.getBytes();
try {
sDriver.write(opt, 15);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}

private void updateReceivedData(byte[] data) {
data[2]=' ';
```

```
data[3]=' ';
data[4]=' ';
data[5]=' ';
data[6]=' ';
String str = new String(data);
onAddItem(str);
}


**********AdministradorEnrolar**********

Button bGuardarEnrolar, bRegresarEnrolar;
EditText eNombre, eIngreso, eSalida, eCodigo, eMateria, eParalelo, eAula;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_administrador_enrolar);
bGuardarEnrolar = (Button) findViewById(R.id.GUARDAR_ENR);
bRegresarEnrolar = (Button) findViewById(R.id.REGRESAR_ENR);
eNombre = (EditText) findViewById(R.id.NOM_EN);
eIngreso = (EditText) findViewById(R.id.HORA_IN);
eSalida = (EditText) findViewById(R.id.HORA_OUT);
eCodigo = (EditText) findViewById(R.id.COD_EN);
eMateria = (EditText) findViewById(R.id.MAT_EN);
eParalelo = (EditText) findViewById(R.id.PARALELO_EN);
eAula = (EditText) findViewById(R.id.AULA_EN);

bGuardarEnrolar.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
byte[] opt;
String se="E0\r";
opt = se.getBytes();
try {
sDriver.write(opt, 15);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

if(eNombre.getText().toString().length()>29){
opt = ("TNombre: "+eNombre.getText().toString().substring(0,29)+"\r").getBytes();
}else{
opt = ("TNombre: "+eNombre.getText().toString()+"\r").getBytes();
}

try {
sDriver.write(opt, 50);
} catch (IOException e) {
// TODO Auto-generated catch block
```

```
e.printStackTrace();
}

if(eIngreso.getText().toString().length()>29){
opt =("TIngreso: "+eIngreso.getText().toString().substring(0,29)+"\r").getBytes();
}else{
opt = ("TIngreso: "+eIngreso.getText().toString()+"\r").getBytes();
}

try {
sDriver.write(opt, 50);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
if(eSalida.getText().toString().length()>29){
opt = ("TSalida: "+eSalida.getText().toString().substring(0,29)+"\r").getBytes();
}else{
opt = ("TSalida: "+eSalida.getText().toString()+"\r").getBytes();
}

try {
sDriver.write(opt, 50);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
if(eCodigo.getText().toString().length()>29){
opt = ("TCódigo: "+eCodigo.getText().toString().substring(0,29)+"\r").getBytes();
}else{
opt = ("TCódigo: "+eCodigo.getText().toString()+"\r").getBytes();
}

try {
sDriver.write(opt, 50);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
if(eMateria.getText().toString().length()>29){
opt =("TMateria: "+eMateria.getText().toString().substring(0,29)+"\r").getBytes();
}else{
opt = ("TMateria: "+eMateria.getText().toString()+"\r").getBytes();
}

try {
sDriver.write(opt, 50);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

```
if(eParalelo.getText().toString().length()>29){
opt=("TParalelo:"+eParalelo.getText().toString().substring(0,29)+"\r").getBytes();
}else{
opt = ("TParalelo: "+eParalelo.getText().toString()+"\r").getBytes();
}

try {
sDriver.write(opt, 50);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
if(eAula.getText().toString().length()>29){
opt = ("TAula: "+eAula.getText().toString().substring(0,29)+"\r").getBytes();
}else{
opt = ("TAula: "+eAula.getText().toString()+"\r").getBytes();
}

try {
sDriver.write(opt, 50);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

showConsoleActivity(6,"COLOCAR DEDO DEL USUARIO");
}
});

bRegresarEnrolar.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
showConsoleActivity(4,"USUARIO NO ENROLADO");
}
});
}

**********AdministradorColocarDedo***********

package com.example.hmi_robotica;
private void updateReceivedData(byte[] data) {

switch (data[1]){
case '0':
showConsoleActivity(5,"HUELLA INCORRECTA");
break;
case '1':
showConsoleActivity(4,"ENROLADO EXITOSO");
break;
default:
break;
```

```
}
}

**********AdministradorExtrInfo*********************

Button bExtraerInfo, bRegresarInfo;
int ext=0;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_administrador_ext_info);

bExtraerInfo = (Button) findViewById(R.id.EXTRAER_INF);
bRegresarInfo = (Button) findViewById(R.id.REGRESAR_INF);

bExtraerInfo.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {

if(ext==0){
byte[] opt;
String se="I0\r";
opt = se.getBytes();
try {
sDriver.write(opt, 15);
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

ext=1;
}
}
});

bRegresarInfo.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
showConsoleActivity(4,"INFORMACIÓN NO EXTRAIDA");
}
});
}
private void updateReceivedData(byte[] data) {
if(data[1]=='0'){
showConsoleActivity(4,"INFORMACIÓN NO EXTRAIDA");
}
if(data[1]=='1'){
showConsoleActivity(4,"INFORMACIÓN EXTRAIDA");
}
}
```

ANEXO C. HOJA DE DATOS DEL FABRICANTE
Especificaciones Técnicas STM32F4 DISCOVERY

life.augmented

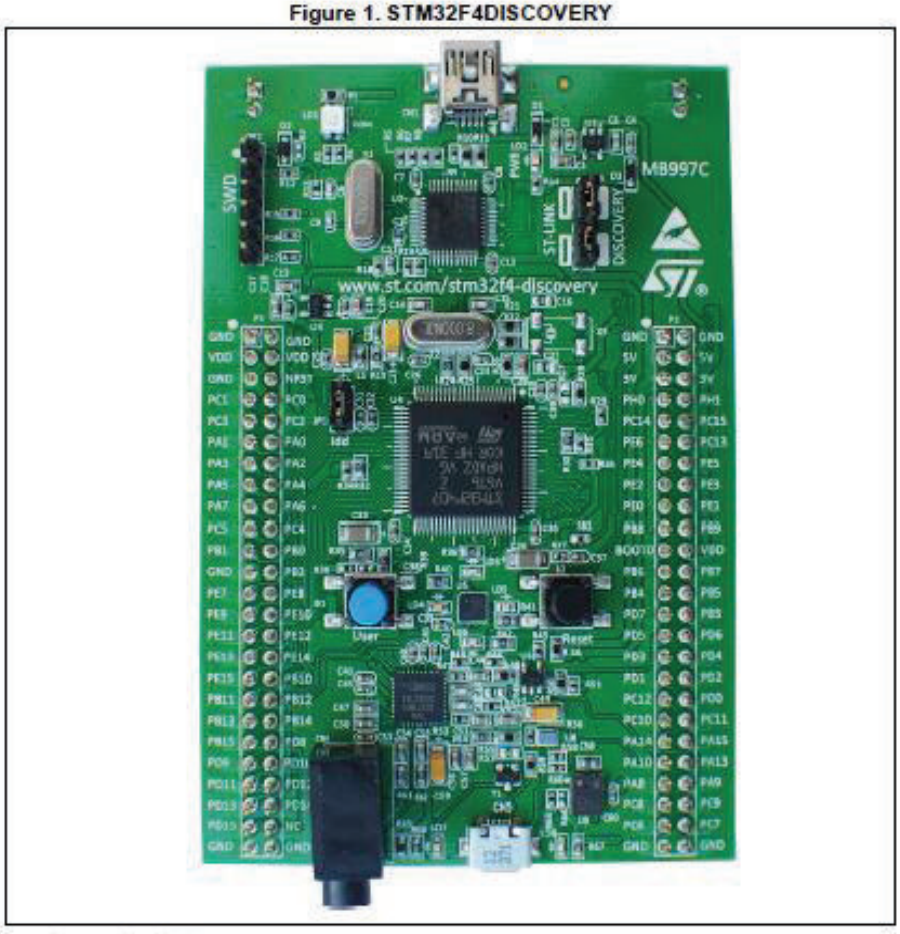
UM1472
User manual

Discovery kit for STM32F407/417 lines

Introduction

The STM32F4DISCOVERY helps you to discover the STM32F407 & STM32F417 lines' high-performance features and to develop your applications.
It is based on an STM32F407VGT6 and includes an ST-LINK/V2 embedded debug tool interface, ST MEMS digital accelerometer, ST MEMS digital microphone, audio DAC with integrated class D speaker driver, LEDs, pushbuttons and a USB OTG micro-AB connector.

Figure 1. STM32F4DISCOVERY



1 Conventions

Table 1 provides the definition of some conventions used in the present document.

Table 1. ON/OFF conventions

Convention	Definition
Jumper JP1 ON	Jumper fitted
Jumper JP1 OFF	Jumper not fitted
Solder bridge SBx ON	SBx connections closed by solder
Solder bridge SBx OFF	SBx connections left open

2 Quick start

The STM32F4DISCOVERY is a low-cost and easy-to-use development kit to quickly evaluate and start a development with an STM32F4 high-performance microcontroller.

Before installing and using the product, please accept the Evaluation Product License Agreement from www.st.com/stm32f4-discovery.

For more information on the STM32F4DISCOVERY and for demonstration software, visit www.st.com/stm32f4-discovery.

2.1 Getting started

Follow the sequence below to configure the STM32F4DISCOVERY board and launch the DISCOVER application:

1. Check jumper position on the board, JP1 on, CN3 on (DISCOVERY selected).
2. Connect the STM32F4DISCOVERY board to a PC with a USB cable 'type A to mini-B' through USB connector CN1 to power the board. Red LED LD2 (PWR) then lights up.
3. Four LEDs between B1 and B2 buttons are blinking.
4. Press user button B1 to enable the ST MEMS sensor, move the board and observe the four LEDs blinking according to the motion direction and speed. (If you connect a second USB cable 'type A to micro-B' between PC and CN5 connector then the board is recognized as standard mouse and its motion will also control the PC cursor).
5. To study or modify the DISCOVER project related to this demo, visit www.st.com/stm32f4-discovery and follow the tutorial.
6. Discover the STM32F4 features, download and execute programs proposed in the list of projects.
7. Develop your own application using available examples.

2.2 System requirements

- Windows PC (XP, Vista, 7)
- USB type A to Mini-B USB cable

2.3 Development toolchain supporting the STM32F4DISCOVERY

- Altium, TASKING™ VX-Toolset
- Atollic TrueSTUDIO®
- IAR Embedded Workbench® for ARM (EWARM)
- Keil™, MDK-ARM

2.4 Order code

To order the STM32F4 high-performance discovery board, use the order code STM32F4DISCOVERY.

3 Features

The STM32F4DISCOVERY offers the following features:

- STM32F407VGT6 microcontroller featuring 1 MB of Flash memory, 192 KB of RAM in an LQFP100 package
- On-board ST-LINK/V2 with selection mode switch to use the kit as a standalone ST-LINK/V2 (with SWD connector for programming and debugging)
- Board power supply: through USB bus or from an external 5V supply voltage
- External application power supply: 3V and 5V
- LIS302DL or LIS3DSH, ST MEMS motion sensor, 3-axis digital output accelerometer
- MP45DT02, ST MEMS audio sensor, omnidirectional digital microphone
- CS43L22, audio DAC with integrated class D speaker driver
- Eight LEDs:
  - LD1 (red/green) for USB communication
  - LD2 (red) for 3.3V power on
  - Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)
  - 2 USB OTG LEDs LD7 (green) VBus and LD8 (red) over-current
- Two pushbuttons (user and reset)
- USB OTG with micro-AB connector
- Extension header for LQFP100 I/Os for quick connection to prototyping board and easy probing

4 Hardware and layout

The STM32F4DISCOVERY is designed around the STM32F407VGT6 microcontroller in a 100-pin LQFP package.

Figure 2 illustrates the connections between the STM32F407VGT6 and its peripherals (ST-LINK/V2, pushbutton, LED, Audio DAC, USB, ST MEMS accelerometer, ST MEMS microphone, and connectors).

Figure 3 and *Figure 4* help you to locate these features on the STM32F4DISCOVERY.

Figure 2. Hardware block diagram

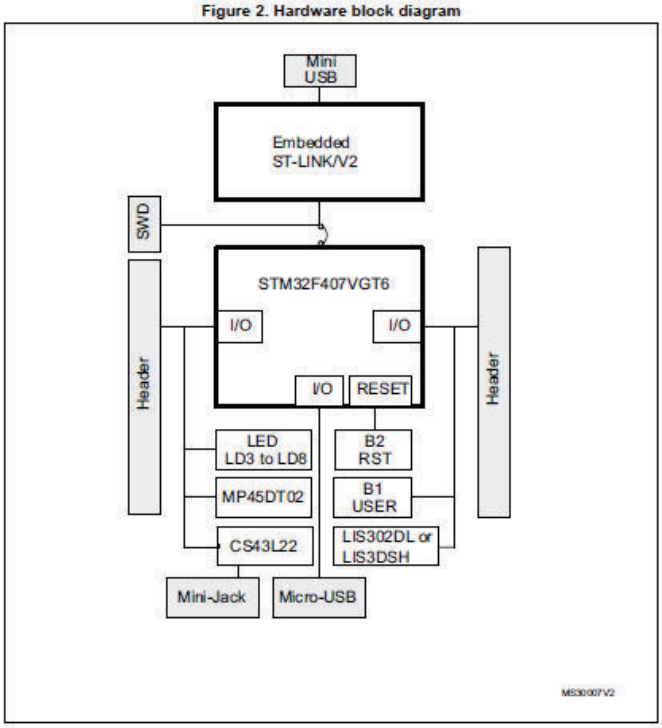
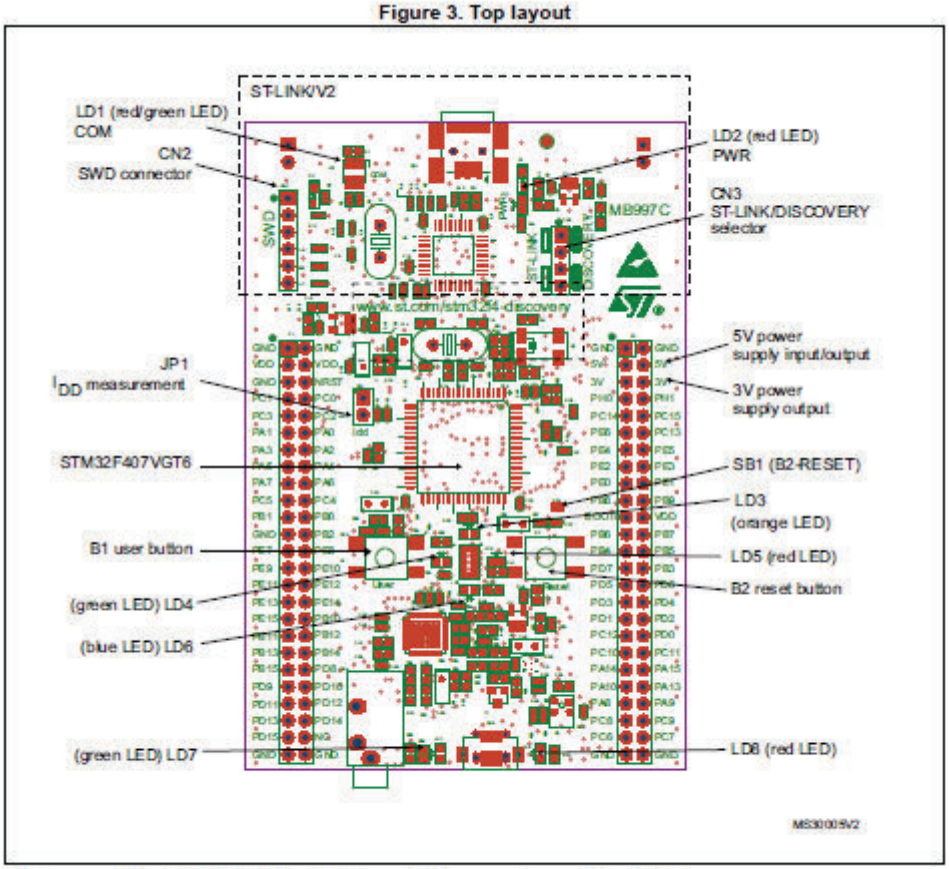


Figure 3. Top layout

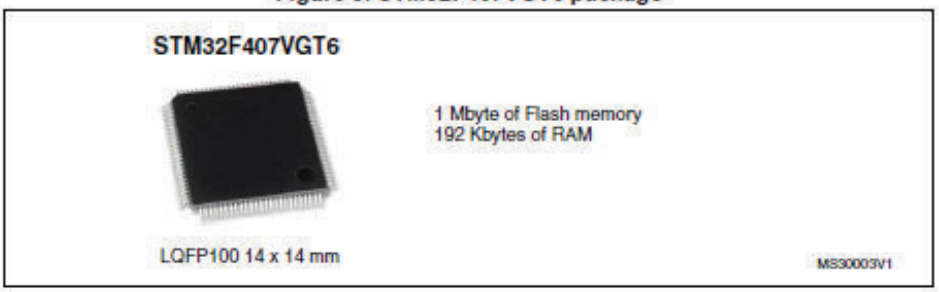


Note: Pin 1 of CN2, CN3, JP1, P1 and P2 connectors are identified by a square.

4.1 STM32F407VGT6 microcontroller

This ARM Cortex-M4 32-bit MCU with FPU has 210 DMIPS, up to 1 MB Flash/192+4 KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces and a camera.

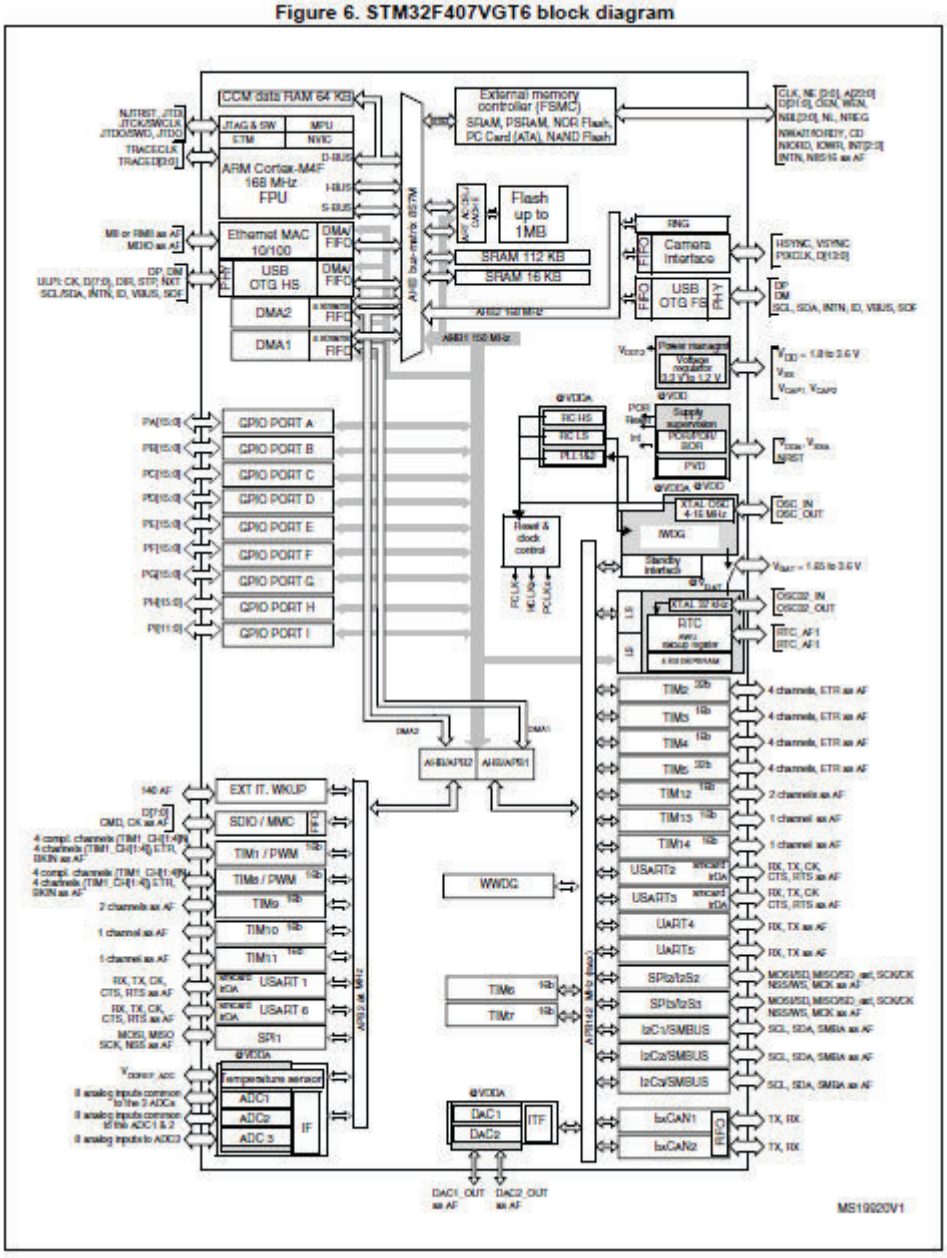
Figure 5. STM32F407VGT6 package



This device provides the following benefits.

- 168 MHz/210 DMIPS Cortex-M4 with single cycle DSP MAC and floating point unit providing:
  - Boosted execution of control algorithms
  - More features possible for your applications
  - Ease of use
  - Better code efficiency
  - Faster time to market
  - Elimination of scaling and saturation
  - Easier support for meta-language tools
- Designed for high performance and ultra fast data transfers; ART Accelerator, 32-bit, 7-layer AHB bus matrix with 7 masters and 8 slaves including 2 blocks of SRAM, Multi DMA controllers: 2 general purpose, 1 for USB HS, 1 for Ethernet, One SRAM block dedicated to the core, providing performance equivalent to 0-wait execution from Flash Concurrent execution and data transfers and simplified resource allocation
- Outstanding power efficiency; Ultra-low dynamic power, RTC <1 µA typical in VBAT mode, 3.6 V down to 1.7 V VDD, Voltage regulator with power scaling capability, providing extra flexibility to reduce power consumption for applications requiring both high processing and low power performance when running at low voltage or on a rechargeable battery
- Maximum integration: Up to 1 Mbyte of on-chip Flash memory, 192 Kbytes of SRAM, reset circuit, internal RCs, PLLs, WLCSP package available, providing more features in space constrained applications
- Superior and innovative peripherals providing new possibilities to connect and communicate high speed data and more precision due to high resolution
- Extensive tools and software solutions providing a wide choice within the STM32 ecosystem to develop your applications.

Figure 6. STM32F407VGT6 block diagram



4.2 Embedded ST-LINK/V2

The ST-LINK/V2 programming and debugging tool is integrated on the STM32F4DISCOVERY. The embedded ST-LINK/V2 can be used in 2 different ways according to the jumper states (see *Table 2*):

- Program/debug the MCU on board,
- Program/debug an MCU in an external application board using a cable connected to SWD connector CN2.

The embedded ST-LINK/V2 supports only SWD for STM32 devices. For information about debugging and programming features refer to user manual UM1075 (*ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32*) which describes in detail all the ST-LINK/V2 features.

Figure 7. Typical configuration

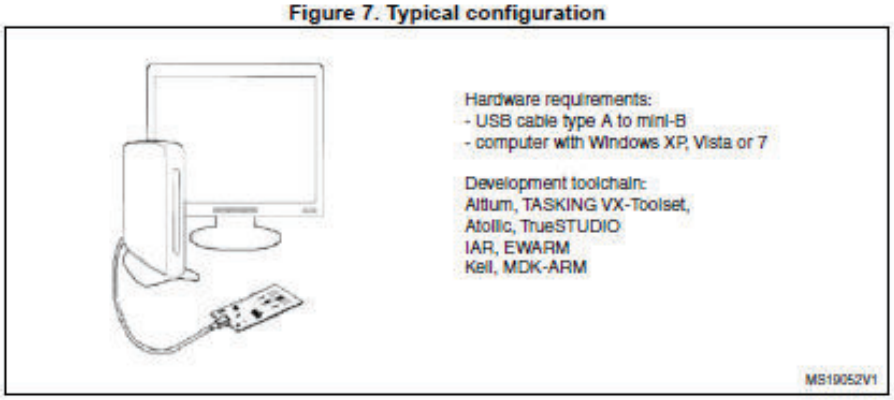


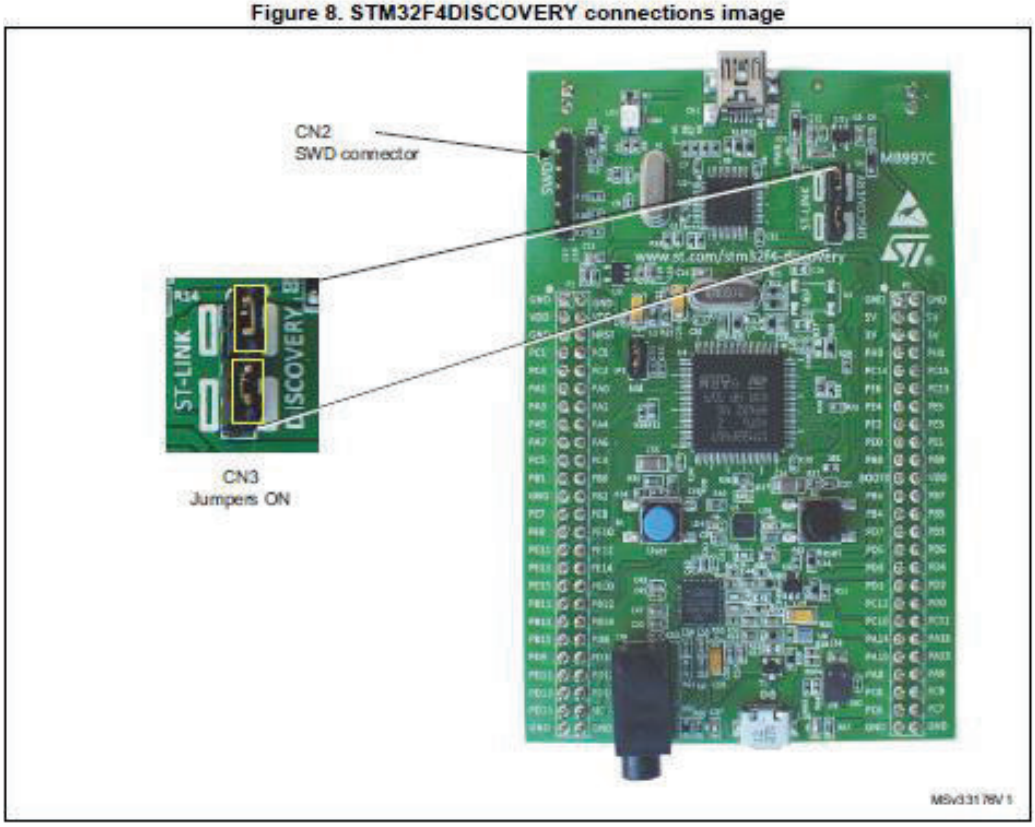
Table 2. Jumper states

Jumper state	Description
Both CN3 jumpers ON	ST-LINK/V2 functions enabled for on board programming (default)
Both CN3 jumpers OFF	ST-LINK/V2 functions enabled for application through external CN2 connector (SWD supported)

4.2.1 Using ST-LINK/V2 to program/debug the STM32F4 on board

To program the STM32F4 on board, simply plug in the two jumpers on CN3, as shown in *Figure 8* in red, but do not use the CN2 connector as that could disturb communication with the STM32F407VGT6 of the STM32F4DISCOVERY.

Figure 8. STM32F4DISCOVERY connections image



4.2.2 Using ST-LINK/V2 to program/debug an external STM32 application

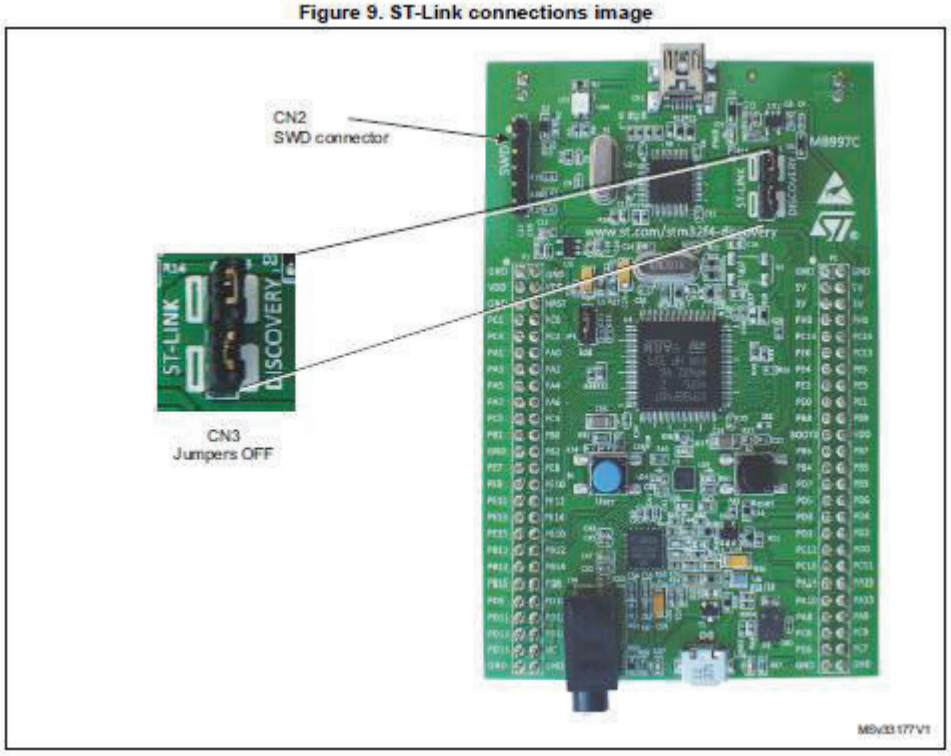
It is very easy to use the ST-LINK/V2 to program the STM32 on an external application. Simply remove the 2 jumpers from CN3 as shown in *Figure 9*, and connect your application to the CN2 debug connector according to *Table 3*.

Note: SB11 must be OFF if you use CN2 pin 5 in your external application.

Table 3. Debug connector CN2 (SWD)

Pin	CN2	Designation
1	VDD_TARGET	VDD from application
2	SWCLK	SWD clock
3	GND	Ground
4	SWDIO	SWD data input/output
5	NRST	RESET of target MCU
6	SWO	Reserved

Figure 9. ST-Link connections image



4.3 Power supply and power selection

The power supply is provided either by the host PC through the USB cable, or by an external 5V power supply.

The D1 and D2 diodes protect the 5V and 3V pins from external power supplies:

- 5V and 3V can be used as output power supplies when another application board is connected to pins P1 and P2.
  In this case, the 5V and 3V pins deliver a 5V or 3V power supply and power consumption must be lower than 100 mA.
- 5V can also be used as input power supplies e.g. when the USB connector is not connected to the PC.
  In this case, the STM32F4DISCOVERY board must be powered by a power supply unit or by auxiliary equipment complying with standard EN-60950-1: 2006+A11/2009, and must be Safety Extra Low Voltage (SELV) with limited power capability.

4.4 LEDs

- LD1 COM: LD1 default status is red. LD1 turns to green to indicate that communications are in progress between the PC and the ST-LINK/V2.
- LD2 PWR: red LED indicates that the board is powered.
- User LD3: orange LED is a user LED connected to the I/O PD13 of the STM32F407VGT6.
- User LD4: green LED is a user LED connected to the I/O PD12 of the STM32F407VGT6.
- User LD5: red LED is a user LED connected to the I/O PD14 of the STM32F407VGT6.
- User LD6: blue LED is a user LED connected to the I/O PD15 of the STM32F407VGT6.
- USB LD7: green LED indicates when VBUS is present on CN5 and is connected to PA9 of the STM32F407VGT6.
- USB LD8: red LED indicates an overcurrent from VBUS of CN5 and is connected to the I/O PD5 of the STM32F407VGT6.

4.5 Pushbuttons

- B1 USER: User and Wake-Up button connected to the I/O PA0 of the STM32F407VGT6.
- B2 RESET: Pushbutton connected to NRST is used to RESET the STM32F407VGT6.

4.6 On board audio capability

The STM32F4 uses an audio DAC (CS43L22) to output sounds through the audio mini jack connector.

The STM32F4 controls the audio DAC through the I2C interface and processes digital signals through I2S connection or analog input signal.

- The sound can come independently from different inputs:
  - ST MEMS microphone (MP45DT02): digital using PDM protocol or analog when using the low pass filter.
  - USB connector: from external mass storage such as a USB key, USB HDD, and so on.
  - Internal memory of the STM32F4.
- The sound can be output in different ways through audio DAC:
  - Using I2S protocol
  - Using the STM32F4 DAC to analog input AIN1x of the CS43L22
  - Using the microphone output directly via a low pass filter to analog input AIN4x of the CS43L22

4.7 USB OTG supported

The STM32F4 is used to drive only USB OTG full speed on this board. The USB micro-AB connector (CN5) allows the user to connect a host or device component, such as a USB key, mouse, and so on.

Two LEDs are dedicated to this module:

- LD7 (green LED) indicates when VBUS is active
- LD8 (red LED) indicates an overcurrent from connected device

4.8 Motion sensor (ST MEMS LIS302DL or LIS3DSH)

Two different versions of motion sensor (U5 in schematic) are available on the board depending the PCB version. The LIS302DL is present on board MB997B (PCB revision B) and the LIS3DSH is present on board MB997C (PCB rev C).

The LIS302DL or LIS3DSH are both an ultra compact low-power three-axis linear accelerometer.

It includes a sensing element and an IC interface able to provide the measured acceleration to the external world through I2C/SPI serial interface.

The LIS302DL has dynamically user selectable full scales of +-2g/+-8g and it is capable of measuring acceleration with an output rate of 100Hz to 400Hz.

The LIS3DSH has ±2g/±4g/±6g/±8g/±16g dynamically selectable full-scale and it is capable of measuring acceleration with an output data rate of 3.125 Hz to 1.6 kHz.

The STM32F4 controls this motion sensor through the SPI interface.

4.9 JP1 (Idd)

Jumper JP1, labeled Idd, allows the consumption of STM32F407VGT6 to be measured by removing the jumper and connecting an ammeter.

- Jumper on: STM32F407VGT6 is powered (default).
- Jumper off: an ammeter must be connected to measure the STM32F407VGT6 current, (if there is no ammeter, the STM32F407VGT6 is not powered).

4.10 OSC clock

4.10.1 OSC clock supply

If PH0 and PH1 are only used as GPIOs instead of as a clock, then SB13 and SB14 are closed and R24, R25 and R68 are removed.

- **MCO from ST-LINK**. From MCO of the STM32F103. This frequency cannot be changed, it is fixed at 8 MHz and connected to PH0-OSC_IN of the STM32F407VGT6. Configuration needed:
  - SB13, SB14 OPEN
  - R25[a] removed
  - R68[a] soldered
- **Oscillator onboard**. From X2 crystal. For typical frequencies and its capacitors and resistors, please refer to the STM32F407VGT6 Datasheet. Configuration needed:
  - SB13, SB14 OPEN
  - R25[a] soldered
  - R68[a] removed
- **Oscillator from external PH0**. From external oscillator through pin 7 of the P2 connector. Configuration needed:
  - SB13 closed
  - SB14 closed
  - R25 and R68 removed

4.10.2 OSC 32 KHz clock supply

If PC14 and PC15 are only used as GPIOs instead of as a clock, then SB15 and SB16 are closed, and R21 and R22 are removed.

- **Oscillator onboard**. From X1 Crystal (not provided). Configuration needed:
  - SB15, SB16 OPEN
  - C16, C27, R21 and R22 soldered.
- **Oscillator from external PC14**. From external oscillator trough the pin 9 of P2 connector. Configuration needed:
  - SB16 closed
  - SB15 closed
  - R21 and R22 removed

4.12 Extension connectors

The male headers P1 and P2 can connect the STM32F4DISCOVERY to a standard prototyping/wrapping board. STM32F407VGT6 GPI/Os are available on these connectors. P1 and P2 can also be probed by an oscilloscope, logical analyzer or voltmeter.

Table 5. MCU pin description versus board function

MCU pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
BOOT0	VPP	94	-	-	-	-	-	-	-	-	-	-	-	-	-	21
NRST	-	14	-	-	-	RESET	-	NRST	-	-	-	-	-	5	6	-
PA0-WKUP	USART2_CTS/ USART4_TX/ ETH_MII_CRS/ TIM2_CH1_ETR/ TIM5_CH1/ TIM8_ETR/ ADC123_IN0/ WKUP	23	-	-	-	USER	-	-	-	-	-	-	-	-	12	-
PA1	USART2_RTS/ USART4_RX/ ETH_RMII_REF_CLK/ ETH_MII_RX_CLK/ TIM5_CH2/ TIMM2_CH2/ ADC123_IN1	24	-	-	-	-	-	-	-	-	-	-	-	-	11	-
PA2	USART2_TX/ TIM5_CH3/ TIM9_CH1/ TIM2_CH3/ ETH_MDIO/ ADC123_IN2	25	-	-	-	-	-	-	-	-	-	-	-	-	14	-

Table 5. MCU pin description versus board function (continued)

MCU pin			Board function													
Main function	Alternate functions	LQFP100	CS 43L22	MP45DT02	LIS302DL or LIS3DSH	Push button	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PA3	USART2_RX/ TIM5_CH4/ TIM9_CH2/ TIM2_CH4/ OTG_HS_ULPI_D0/ ETH_MII_COL/ ADC123_IN3	26	-	-	-	-	-	-	-	-	-	-	-	-	13	-
PA4	SPI1_NSS/ SPI3_NSS/ USART2_CK/ DCMI_HSYNC/ OTG_HS_SOF/ I2S3_WS/ ADC12_IN4/ DAC1_OUT	29	LRCK/AIN1x	-	-	-	-	-	-	-	-	-	-	-	16	-
PA5	SPI1_SCK/ OTG_HS_ULPI_CK/ TIM2_CH1_ETR/ TIM8_CHIN/ ADC12_IN5/ DAC2_OUT	30	-	-	SCL/SPC	-	-	-	-	-	-	-	-	-	15	-
PA6	SPI1_MISO/ TIM8_BKIN/ TIM13_CH1/ DCMI_PIXCLK/ TIM3_CH1/ TIM1_BKIN/ ADC12_IN6	31	-	-	SDO	-	-	-	-	-	-	-	-	-	18	-
PA7	SPI1_MOSI/ TIM8_CH1N/ TIM14_CH1TIM3_CH2/ ETH_MII_RX_DV/ TIM1_CH1N/ RMII_CRS_DV/ ADC12_IN7	32	-	-	SDA/SDI/SDO	-	-	-	-	-	-	-	-	-	17	-

Table 5. MCU pin description versus board function (continued)

MCU pin			Board function														
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Push button	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PA8	MCO1/ USART1_CK/ TIM1_CH1/ I2C3_SCL/ OTG_FS_SOF	67	-	-	-	-	-	-	-	-	-	-	-	-	-	43
PA9	USART1_TX/ TIM1_CH2/ I2C3_SMBA/ DCMI_D0/ OTG_FS_VBUS	68	-	-	-	-	GREEN	-	VBUS	-	-	-	1	-	-	44
PA10	USART1_RX/ TIM1_CH3/ OTG_FS_ID/ DCMI_D1	69	-	-	-	-	-	-	ID	-	-	-	4	-	-	41
PA11	USART1_CTS/ CAN1_RX/ TIM1_CH4/ OTG_FS_DM	70	-	-	-	-	-	-	DM	-	-	-	2	-	-	-
PA12	USART1_RTS/ CAN1_TX/ TIM1_ETR/ OTG_FS_DP	71	-	-	-	-	-	-	DP	-	-	-	3	-	-	-
PA13	JTMS-SWDIO	72	-	-	-	-	-	SWDIO	-	-	-	-	-	4	-	42
PA14	JTCK-SWCLK	76	-	-	-	-	-	SWCLK	-	-	-	-	-	2	-	39
PA15	JTDI/ SPI3_NSS/ I2S3_WS/ TIM2_CH1_ETR/ SPI1_NSS	77	-	-	-	-	-	-	-	-	-	-	-	-	-	40

Table 5. MCU pin description versus board function (continued)

MCU pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PH1	OSC_OUT	13	-	-	-	-	-	-	-	OSC_OUT	-	-	-	-	-	8
-	-	-	-	-	-	-	-	-	-	-	-	5V	-	-	-	3
-	-	-	-	-	-	-	-	-	-	-	-	5V	-	-	-	4
-	-	-	-	-	-	-	-	-	-	-	-	3V	-	-	-	5
-	-	-	-	-	-	-	-	-	-	-	-	3V	-	-	-	6
-	-	-	-	-	-	-	-	-	-	-	-	VDD	-	-	3	-
-	-	-	-	-	-	-	-	-	-	-	-	VDD	-	-	4	-
-	-	-	-	-	-	-	-	-	-	-	-	VDD	-	-	-	22
-	-	-	-	-	-	-	-	GND	GND	-	-	GND	5	3	1	-
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	2	-
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	5	-
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	23	-
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	49	-
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	50	-
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	-	1
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	-	2

Table 5. MCU pin description versus board function (continued)

MCU pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	-	49
-	-	-	-	-	-	-	-	-	-	-	-	GND	-	-	-	50

6 Electrical schematics

Figure 11. STM32F4DISCOVERY

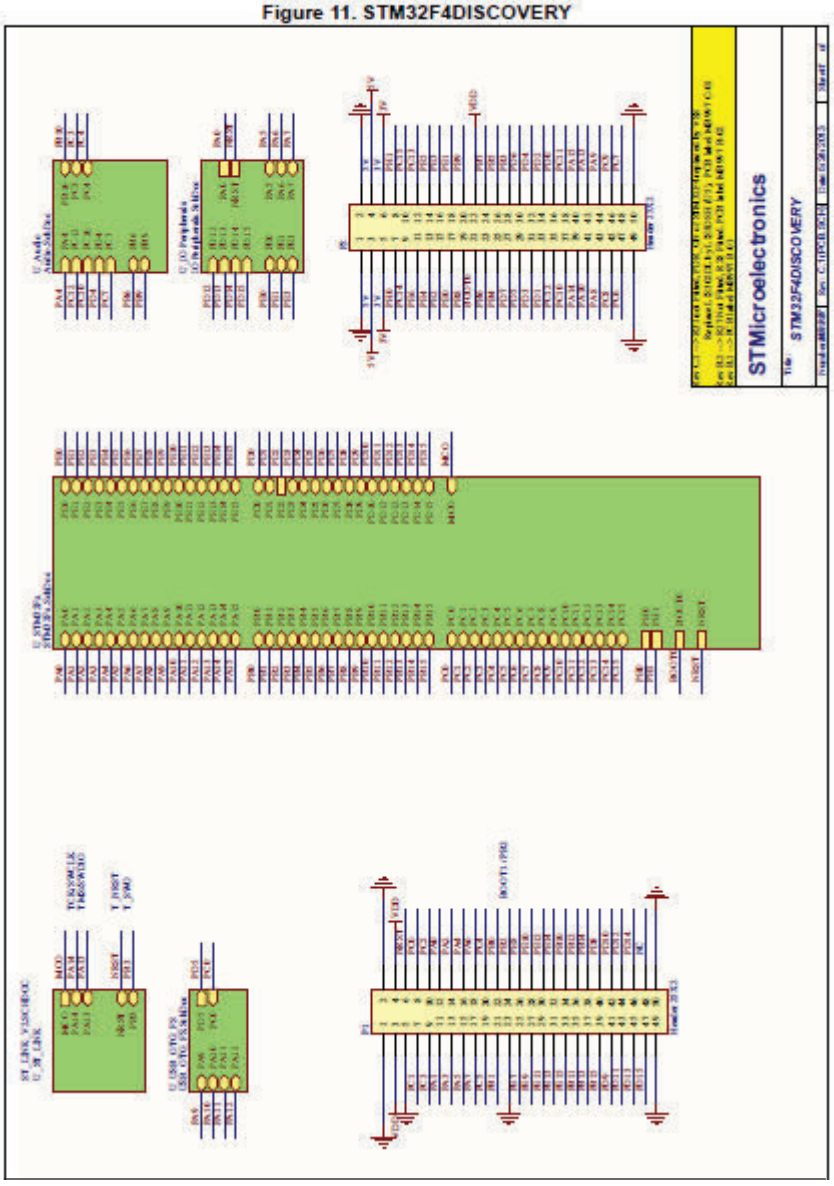


Figure 12. ST-LINK/V2 (SWD only)

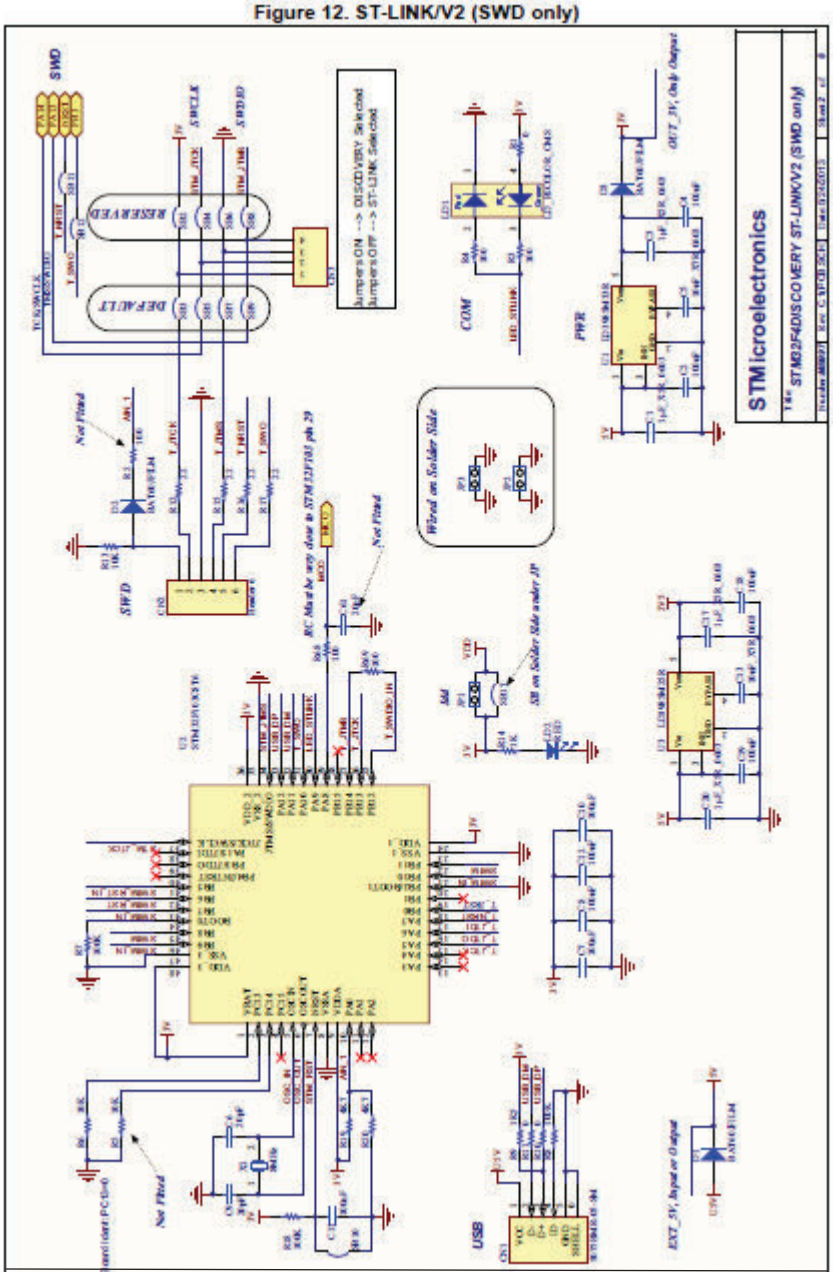
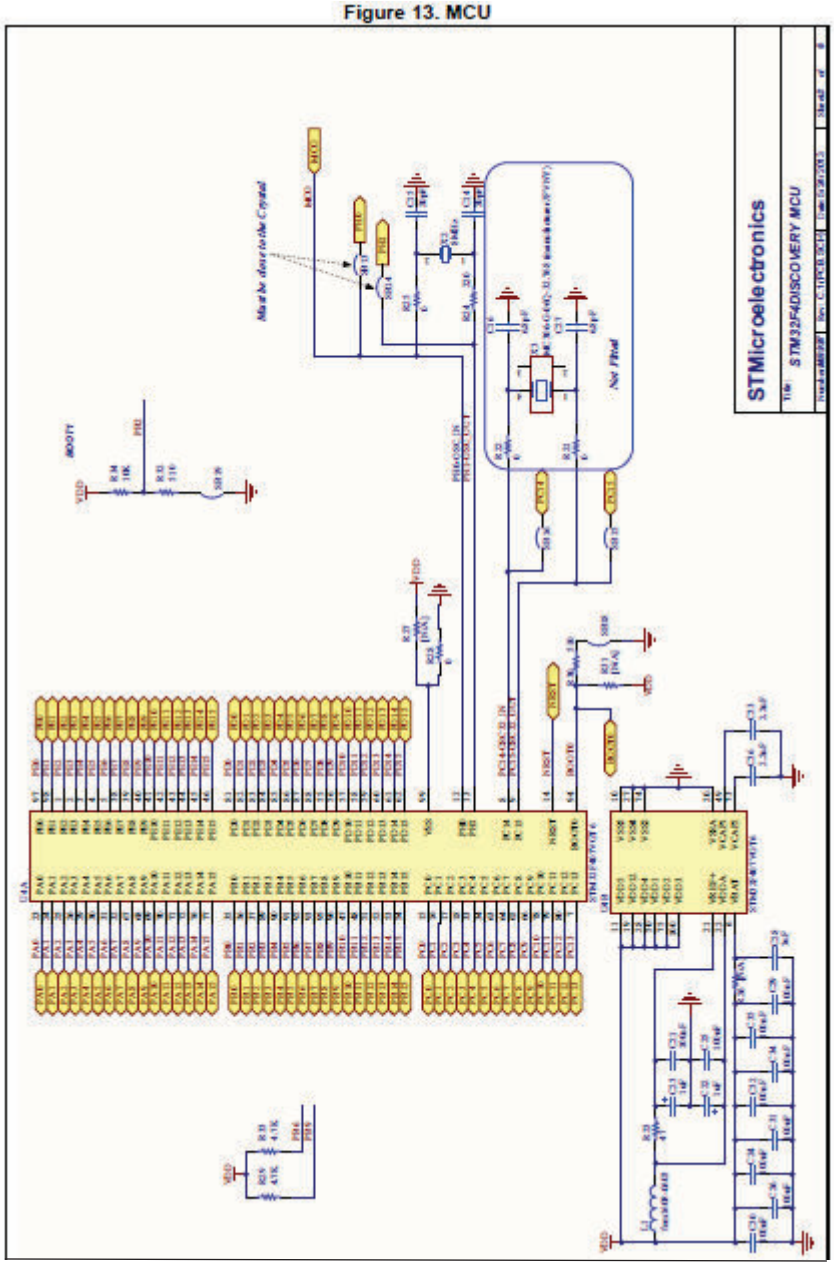


Figure 13. MCU



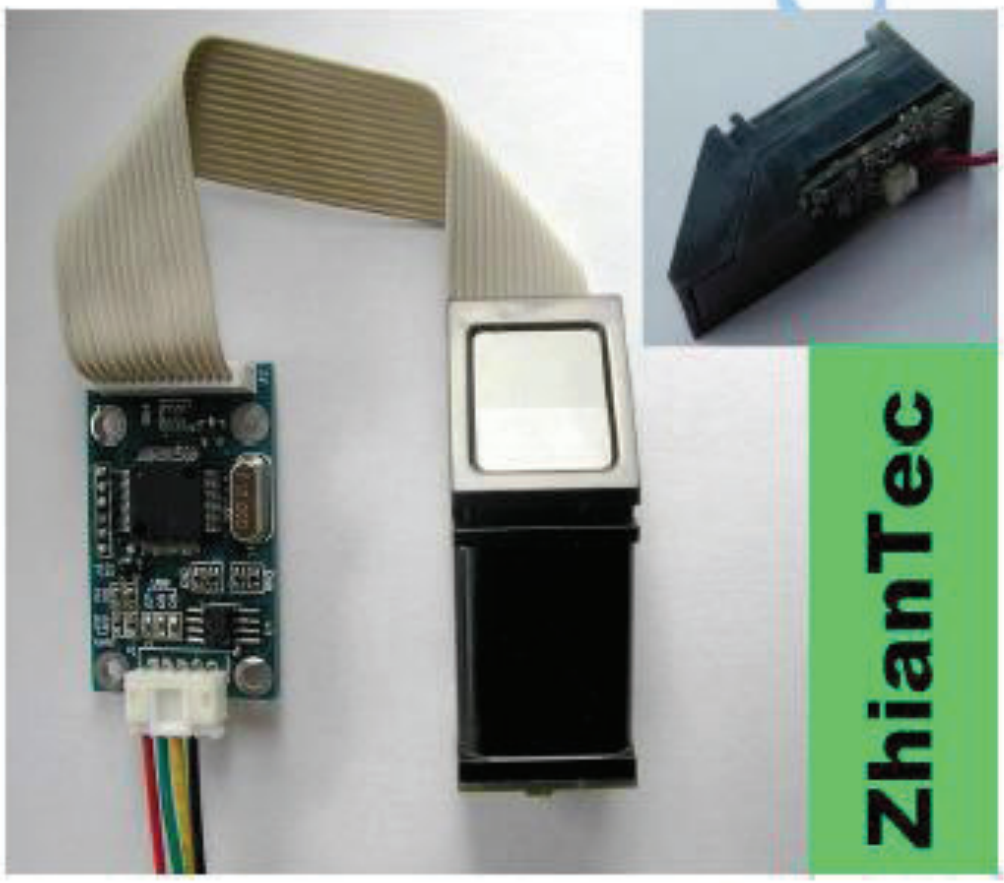
ANEXO D. HOJA DE DATOS DEL FABRICANTE

Especificaciones Técnicas ZMF-20 SERIES FINGER

ZHIANTEC

ZFM-20 Series
Fingerprint Identification Module

User Manual



Hangzhou Zhian Technologies Co., Ltd

Sep 2008 Ver: 1.4

I Introduction

ZFM-20 series are separate fingerprint identification modules proposed by Hangzhou Zhian Technologies Co., Ltd., which takes Synochip DSP as the main processor and optical sensor with Zhian's own intellectual property rights. The module performs series of functions like fingerprint enrollment, image processing, fingerprint matching, serching and template storage.

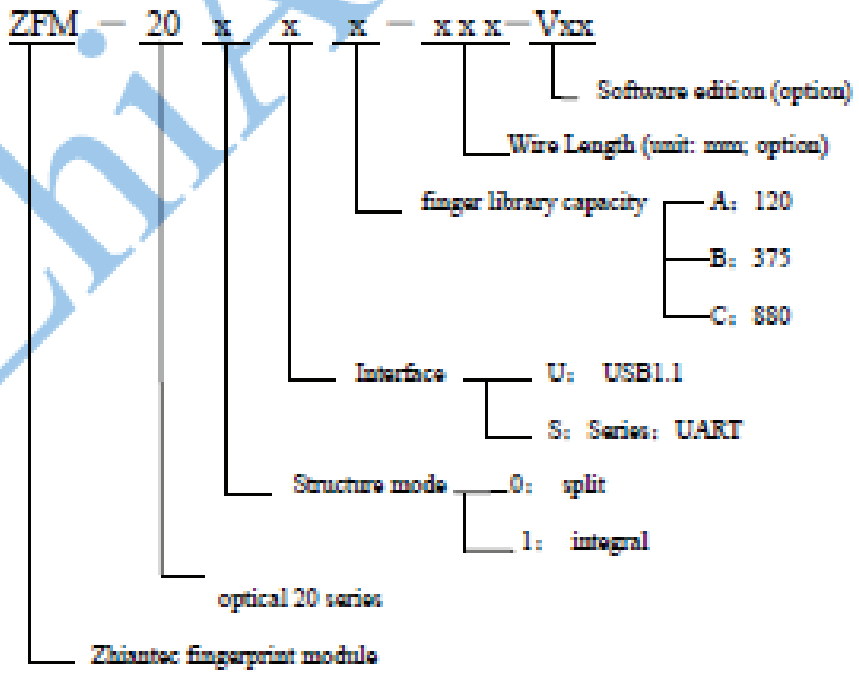
1.1 Operation Principle

Fingerprint processing includes two parts: fingerprint enrollment and fingerprint matching (the matching can be 1:1 or 1:N).

When enrolling, user needs to enter the finger two times. The system will process the two time finger images, generate a template of the finger based on processing results and store the template. When matching, user enters the finger through optical sensor and system will generate a template of the finger and compare it with templates of the finger library. For 1:1 matching, system will compare the live finger with specific template designated in the Module; for 1:N matching, or searching, system will search the whole finger library for the matching finger. In both circumstances, system will return the matching result, success or failure.

1.2 Order Information

Naming of our fingerprint modules follows the following rule. When placing order with us, please fill the correct the type name, so that we can provide better service.



Note: 1) Wire length means the length of parallel wire which connects optical sensor and main board. It is only for split mode.

2) Software edition can be omitted in first order or neglected at all. By default, we take it as the latest edition.

II Main Parameters

Power	DC 3.6V-6.0V	**Interface**	UART(TTL logical level)/ USB 1.1
Working current	Typical: 100mA Peak: 150mA	**Matching Mode**	1:1 and 1:N
Baud rate	(9600*N)bps, N=1～12 (default N=6)	**Character file size**	256 bytes
Image acquiring time	<1s	**Template size**	512 bytes
Storage capacity	120/ 375/ 880	**Security level**	5 (1, 2, 3, 4, 5(highest))
FAR	<0.001%	**FRR**	<0.1%
Average searching time	< 1s (1:880)	**Window dimension**	14mm*18mm
Working environment	Temp: -10℃- +40℃	**Storage environment**	Temp: -40℃- +85℃
	RH: 40%-85%		RH: <85%
Outline Dimention	Split type	Module: 42*25*8.5mm (install dimension: 31.5*19mm) Sensor:56*20*21.5mm	
	Integral type	56*20*21.5mm	

III Hardware Interface

3.1 Connecting with upper computer (J1 on board)

Whether the interface is UART or USB (hardware setting is different when out of factory, please don;t misuse), on PCB board the connector is uniform. For split type, 5-pin connector (J1) with 2.0mm space between; for integral type, 4-pin connector (J1) with 1.27mm space between.

Unless required specially by user, the connecting cable with upper computer is 150mm.

3.1.1 Serial Communication

When the FP module communicates with user device, definition of J1 is as follows:

Pin Number	Name	Type	Function Description
1	Vin	in	Power input（cable color: red）
2	TD	out	Data output. TTL logical level (cable color: green)
3	RD	in	Data input. TTL logical level (cable color: white)
4	GND	—	Signal ground. Connected to power ground (cable color: black)
5	NC	—	Not connect. (doesn;t exist with integral type)

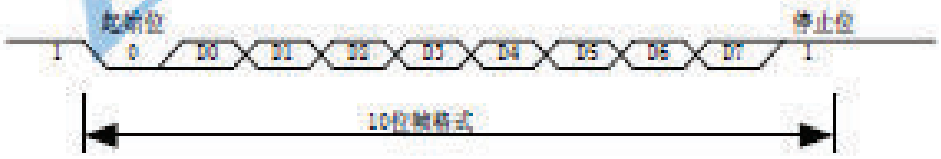
3.1.1.1 Hardware connection

Via serial interface, the Module may communicate with MCU of 3.3V or 5V power: TD (pin 2 of J1) connects with RXD (receiving pin of MCU), RD (pin 3 of J1) connects with TXD (transferring pin of MCU). Should the upper computer (PC) be in RS-232 mode, please add level converting circuit, like MAX232, between the Module and PC.

3.1.1.2 Serial communication protocol

The mode is semiduplex asynchronism serial communication. And the default baud rate is 57600bps. User may set the baud rate in 9600~115200bps.

Transferring frame format is 10 bit: the low-level starting bit, 8-bit data with the LSB first, and an ending bit. There is no check bit.



3.1.1.3 Reset time

At power on, it takes about 500ms for initialization. During this period, the Module can;t accept commands for upper computer.

3.1.1.4 Electrical paramenter (All electrical level takes GND as reference)

1. Power supply

Item	Parameter			Unit	Note
	Min	Typ	Max		
Power Voltage (Vin)	3.6		6.0	V	Normal working value.
Maximum Voltage (Vin_{max})	−0.3		7.0	V	Exceeding the Maximum rating may cause permant harm to the Module.
Operation Current (Icc)	90	100	110	mA	
Peak Current (Ipeak)			150	mA	

2. TD（output, TTL logic level）

Item	Condition	Parameter			Unit	Note
		Min	Typ	Max		
V_{OL}	I_{OL}= −4mA			0.4	V	Logic 0
V_{OH}	I_{OH}= 4mA	2.4		3.3	V	Logic 1

3. RD（input, TTL logic level）

Item	Condition	Parameter			Unit	Note
		Min	Typ	Max		
V_{IL}				0.6	V	Loigc 0
V_{IH}		2.4			V	Logic 1
I_{IH}	V_{IH}=5V		1		mA	
	V_{IH}=3.3V		30		uA	
V_{imax}		−0.3		5.5	V	Maximum input voltage

3.1.2 USB communication

When it's USB communication, definition of J1 is:

Pin Num	Name	Type	Function Description
1	Vin	in	Power supply input（refer to 3.1.1.4 for electrical parameter）
2	DP+	In/Out	USB data.
3	DP-	In/Out	USB data.
4	GND	—	Signal ground. Connected to power ground.
5	END	—	Earth Ground. Float or connect to the shield layer of cable. (doesn't exist with Integral type)

3.2 Connecting with sensor (J2 on board)

15-pin single-row connector (J2) (connecting a 15-wire flat cable) with 1.25mm space between, serves as the connector between the main board and the optical sensor. Unless as specially required by order, the default cable length is 150mm.

For integral type, user needn't to worry, it's connected internally.

IV System Resources

To address demands of different customer, Module system provides abundant resources at user¡¯s use.

4.1 Notepad

The system sets aside a 512-bytes memory (16 pages* 32 bytes) for user¡¯s notepad, where data requiring power-off protection can be stored. The host can access the page by instructions of PS_WriteNotepad and PS_ReadNotepad.

Note: when write on one page of the pad, the entire 32 bytes will be written in wholly covering the original contents.

4.2 Buffer

There are an image buffer and two 512-byte-character-file buffer within the RAM space of the module. Users can read & write any of the buffers by instructions.

Note: Contents of the above buffers will be lost at power-off.

4.2.1 Image buffer

ImageBuffer serves for image storage and the image format is 256*288 pixels.

When transferring through UART, to quicken speed, only the upper 4 bits of the pixel is transferred (that is 16 grey degrees). And two adjacent pixels of the same row will form a byte before the transferring. When uploaded to PC, the 16-grey-degree image will be extended to 256-grey-degree format. That¡¯s 8-bit BMP format.

When transferring through USB, the image is 8-bit pixel, that¡¯s 256 grey degrees.

4.2.2 Character file buffer

Character file buffer, CharBuffer1, CharBuffer2, can be used to store both character file and template file.

4.3 Fingerprint Library

Syustem sets aside a certain space within Flash for fingerprint template storage, that¡¯s fingerprint library. Contents of the library remain at power off.

Capacity of the library changes with the capacity of Flash, system will recognize the latter automatically. Fingerprint template¡¯s storage in Flash is in sequential order. Assume the fingerprint capacity N, then the serial number of template in library is 0, 1, 2, 3 ¡ N. User can only access library by template number.

4.4 System Confiuration Parameter

To facilitate user¡¯s developing, Module opens part system parameters for use. And the basic instructions are SetSysPara & ReadSysPara. Both instructions take Parameter Number as parameter.

When upper computer sends command to modify parameter, Module first responses with original configurations, then performs the parameter modification and writes configuration record into Flash. At the next startup, system will run with the new configurations.

4.4.1 Baud rate control (Parameter Number: 4)

The Parameter controls the UART communication speed of the Modul. Its value is an integer N, N= [1, 12]. Coorresponding baud rate is 9600*N bps.

4.4.2 Security Level (Parameter Number: 5)

The Parameter controls the matching threshold value of fingerprint searching and matching. Security level is divided into 5 grades, and coorresponding value is 1, 2, 3, 4, 5. At level 1, FAR is the highest and FRR is the lowest; however at level 5, FAR is the lowest and FRR is the highest.

4.4.3 Data package length (Parameter Number: 6)

The parameter decides the max length of the transferring data package when communicating with upper computer. Its value is 0, 1, 2, 3, corresponding to 32 bytes, 64 bytes, 128 bytes, 256 bytes respectively.

4.5 System status register

System status register indicates the current operation status of the Module. Its length is 1 word, and can be read via instruction *ReadSysPara*. Definition of the register is as follows:

Bit Num	15 ... 4	3	2	1	0
Description	Reserved	ImgBufStat	PWD	Pass	Busy

Note:

- Busy：1 bit. 1: system is executing commands; 0: system is free;
- Pass：1 bit. 1: find the matching finger; 0: wrong finger;
- PWD：1 bit. 1: Verified device's handshaking password.
- ImgBufStat：1 bit. 1: image buffer contains valid image.

4.6 Module address

Each module has an identifying address. When communicating with upper computer, each instruction/data is transferred in data package form, which contains the address item. Module system only responds to data package whose address item value is the same with its identifying address.

The address length is 4 bytes, and its default factory value is 0xFFFFFFFF. User may modify the address via instruction *SetAddor*. The new modified address remains at power off.

4.7 Random number generator

Module integrates a hardware 32-bit random number generator (RNG) (without seed). Via instruction *GetRandomCode*, system will generate a random number and upload it.

V Communication Protocol

The protocol defines the data exchanging format when ZFM-20 series communicates with upper computer. The protocol and instruction sets applies for both UART and USB communication mode. For PC, USB interface is strongly recommended to improve the exchanging speed, especially in fingerprint scanning device.

5.1 Data package format

When communicating, the transferring and receiving of command/data/result are all wrapped in data package format.

Data package format

Header	Adder	Package identifier	Package length	Package content (instruction/data/Parameter)	Checksum

Definition of Data package

Name	Symbol	Length	Description	
Header	Start	2 bytes	Fixed value of EF01H; High byte transferred first.	
Adder	ADDER	4 bytes	Default value is 0xFFFFFFFF, which can be modified by command. High byte transferred first and at wrong adder value, module will reject to transfer.	
Package identifier	PID	1 byte	01H	Command packet;
			02H	Data packet; Data packet shall not appear alone in executing processs, must follow command packet or acknowledge packet.
			07H	Acknowledge packet;
			08H	End of Data packet.
Package length	LENGTH	2 bytes	Refers to the length of package content (command packets and data packets) plus the length of Checksum(2 bytes). Unit is byte. Max length is 256 bytes. And high byte is transferred first.	
Package contents	DATA	–	It can be commands, data, command's parameters, acknowledge result, etc. (fingerprint character value, template are all deemed as data);	
Checksum	SUM	2 bytes	The arithmetic sum of package identifier, package length and all package contens. Overflowing bits are omitted. high byte is transferred first.	

5.2 Check and acknowledgement of data package

Note: Commands shall only be sent from upper computer to the Module, and the Module

acknowledges the commands.

Upon receipt of commands, Module will report the commands execution status and results to upper computer through acknowledge packet. Acknowledge packet has parameters and may also have following data packet. Upper computer can¡t ascertain Module¡s package receiving status or command execution results unless through acknowledge packet sent from Module. Acknowledge packet includes 1 byte confirmation code and maybe also the returned parameter.

Confirmation code¡s definition is :

1. 00h: commad execution complete;
2. 01h: error when receiving data package;
3. 02h: no finger on the sensor;
4. 03h: fail to enroll the finger;
5. 06h: fail to generate character file due to the over-disorderly fingerprint image;
6. 07h: fail to generate character file due to lackness of character point or over-smallness of fingerprint image
7. 08h: finger doesn¡t match;
8. 09h: fail to find the matching finger;
9. 0Ah: fail to combine the character files;
10. 0Bh: addressing PageID is beyond the finger library;
11. 0Ch: error when reading template from library or the template is invalid;
12. 0Dh: error when uploading template;
13. 0Eh: Module can¡t receive the following data packages.
14. 0Fh: error when uploading image;
15. 10h: fail to delete the template;
16. 11h: fail to clear finger library;
17. 15h: fail to generate the image for the lackness of valid primary image;
18. 18h: error when writing flash;
19. 19h: No definition error;
20. 1Ah: invalid register number;
21. 1Bh: incorrect configuration of register;
22. 1Ch: wrong notepad page number;
23. 1Dh: fail to operate the communication port;
24. others: system reserved;

VI Module Instruction System

ZFM-20 series provide 23 instructions. Through combination of different instructions, application program may realize multi finger authentication functions. All commands/data are transferred in package format. *Refer to 5.1 for the detailed information of package.*

6.1 System-related instructions

6.1.1 Communicate link: handshake

Description:

Confirm that communicate is connect between module and upper monitor

Input Parameter: control code 0

Return Parameter: confirmation code;

Instruction code: 17H

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	1byte	2 bytes
Header	Chip address	Package identifier	Package length	Instruction code	Control code	Checksum
EF01H	xxxx	01H	0004H	17H	0	001CH

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Chip address	Package identifier	Package length	Confirmation code	Checksum
EF01H	xxxx	07H	0003H	xxH	sum

Note: Confirmation code=00H: Port operation complete;
Confirmation code=01H: error when receiving package;
Confirmation code=1dH: fail to operate the communication port;

6.1.2 Set Module address: SetAdder

Description: Set Module address. (Refer to 4.7 for adderss information)

Input Parameter: None;

Return Parameter: Confirmation code (1 byte)

Instruction code: 15H

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	4 bytes	2 bytes
Header	Original Module address	Package identifier	Package length	Instruction code	New Module address	Checksum
EF01H	xxxx	01H	0007H	15H	xxxx	sum

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	New Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	xxxx	0007H	07H	xxH	Sum

Note: Confirmation code=00H: address setting complete;
Confirmation code=01H: error when receiving package;

6.1.3 Set module system¡s basic parameter: SetSysPara

Description: Operation parameter settings. (Refer to 4.4 for more information)
Input Parameter: Parameter number;
Return Parameter: Confirmation code (1 byte)
Instruction code: 0eH
Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	1byte	1byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Parameter number	Contents	Checksum
EF01H	xxxx	01H	0005H	0eH	4/5/6	xx	sum

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	xxxx	0007H	03H	xxH	Sum

Note: Confirmation code=00H: parameter setting complete;
Confirmation code=01H: error when receiving package;
Confirmation code=1aH: wrong register number;

6.1.4 Read system Parameter: ReadSysPara

Description: Read Module¡s status register and system basic configuration parameters; (Refer to 4.4 for system configuration parameter and 4.5 for system status register) .
Input Parameter: none
Return Parameter: Confirmation code (1 byte) + basic parameter (16bytes)
Instruction code: 0fH
Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Checksum
EF01H	xxxx	01H	0003H	0fH	0013H

- Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	16 bytes	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Basic parameter list	Checksum
EF01H	xxxx	07H	3+16 (0013H)	xxH	See following table	sum

Note: Confirmation code=00H: read complete;
Confirmation code=01H: error when receiving package;

Name	Description	Offset (word)	Size (word)
Status register	Contents of system status register	0	1
System identifier code	Fixed value: 0x0009	1	1

Finger library size	Finger library size	2	1
Security level	Security level (1, 2, 3, 4, 5)	3	1
Device address	32-bit device address	4	2
Data packet size	Size code (0, 1, 2, 3)	6	1
Baud settings	N (baud = 9600*N bps)	7	1

6.1.5 Read valid template number: TempleteNum

Description: read the current valid template number of the Module
Input Parameter: none
Return Parameter: Confirmation code (1 byte)，template number:N
Instuction code: 1dH
Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Checksum
EF01H	xxxx	01H	0003H	1DH	0021H

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Template number	Checksum
EF01H	xxxx	07H	5	xxH	N	sum

Note: Confirmation code=00H: read complete;
Confirmation code=01H: error when receiving package;

6.2 Fingerprint-processing instructions

6.2.1 To collect finger image: GenImg

Description: detecting finger and store the detected finger image in ImageBuffer while returning successfull confirmation code; If there is no finger, returned confirmation code would be ; can¡t detect finger¡ .
Input Parameter: none
Return Parameter: Confirmation code (1 byte)
Instuction code: 01H
Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Checksum
EF01H	Xxxx	01H	0003H	01H	0005H

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	Xxxx	07H	0003H	xxH	Sum

Note: Confirmation code=00H: finger collection success;
Confirmation code=01H: error when receiving package;
Confirmation code=02H: can;t detect finger;
Confirmation code=03H: fail to collect finger;

6.2.2 Upload image: UpImage

Description: to upload the image in Img_Buffer to upper computer. Refer to 1.1.1 for more about image buffer.
Input Parameter: none
Return Parameter: Confirmation code (1 byte)
Instuction code: 0aH
Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Checksum
EF01H	Xxxx	01H	0003H	0AH	000EH

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	Xxxx	07H	0003H	xxH	sum

Note 1: Confirmation code=00H: ready to transfer the following data packet;
Confirmation code=01H: error when receiving package;
Confirmation code=0fH: fail to transfer the following data packet;
2: Module shall transfer the following data packet after responding to the upper computer.

6.2.3 Download the image: DownImage

Description: to download image from upper computer to Img_Buffer. Refer to 1.1.1 for more about the image buffer.
Input Parameter: none
Return Parameter: Confirmation code (1 byte)
Instuction code: 0bH
Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Checksum
EF01H	Xxxx	01H	0003H	0bH	000FH

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	Xxxx	07H	0003H	xxH	sum

Note: 1: Confirmation code=00H: ready to transfer the following data packet;

Confirmation code=01H: error when receiving package;
Confirmation code=0eH: fail to transfer the following data packet;

2：Module shall transfer the following data packet after responding to the upper computer. Data package length must be 64, 128, or 256.

6.2.4 To generate character file from image: Img2Tz

Description: to generate character file from the original finger image in ImageBuffer and store the file in CharBuffer1 or CharBuffer2.

Input Parameter: BufferID (character file buffer number)

Return Parameter: Confirmation code (1 byte)

Instruction code: 02H

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Buffer number	Checksum
EF01H	xxxx	01H	0004H	02H	BufferID	sum

Note: BufferID of CharBuffer1 and CharBuffer2 are 1h and 2h respectively. Other values (except 1h, 2h) would be processed as CharBuffer2.

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	xxxx	07H	0003H	XxH	sum

Note: Confirmation code=00H: generate character file complete;
Confirmation code=01H: error when receiving package;
Confirmation code=06H: fail to generate character file due to the over-disorderly fingerprint image;
Confirmation code=07H: fail to generate character file due to lackness of character point or over-smallness of fingerprint image;
Confirmation code=15H: fail to generate the image for the lackness of valid primary image;

6.2.5 To generate template: RegModel

Description: To combine information of character files from CharBuffer1 and CharBuffer2 and generate a template which is stroed back in both CharBuffer1 and CharBuffer2.

Input Parameter：none

Return Parameter：Confirmation code (1 byte)

Instruction code: 05H

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Checksum
EF01H	xxxx	01H	0003H	05H	09H

- Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	xxxx	07H	0003H	xxH	sum

Note: Confirmation code=00H: operation success;
Confirmation code=01H: error when receiving package;
Confirmation code=0aH: fail to combine the character files. That;s, the character files don;t belong to one finger.

6.2.6 To upload character or template: UpChar

Description: to upload the character file or template of CharBuffer1/CharBuffer2 to upper computer;
Input Parameter: BufferID (Buffer number)
Return Parameter: Confirmation code (1 byte)
Instruction code: 08H
Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Buffer number	Checksum
EF01H	xxxx	01H	0004H	08H	BufferID	sum

Note: BufferID of CharBuffer1 and CharBuffer2 are 1h and 2h respectively. Other values (except 1h, 2h) would be processed as CharBuffer2.

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	xxxx	07H	0003H	xxH	sum

Note 1: Confirmation code=00H: ready to transfer the following data packet;
Confirmation code=01H: error when receiving package;
Confirmation code=0dH: error when uploading template;
2: Module shall transfer following data packet after responding to the upper computer.;
3: The instruction doesn;t affect buffer contents.

6.2.7 To download character file or template: DownChar

Description: to download character file or template from upper computer to the specified buffer of Module;
Input Parameter: BufferID (buffer number)
Return Parameter: Confirmation code (1 byte)
Instuction code: 09H
Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	buffer number	Checksum

EF01H	xxxx	01H	0004H	09H	BufferID	sum

Note: BufferID of CharBuffer1 and CharBuffer2 are 1h and 2h respectively. Other values (except 1h, 2h) would be processed as CharBuffer2.

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	xxxx	07H	0003H	xxH	sum

Note 1: Confirmation code=00H: ready to transfer the following data packet;
Confirmation code=01H: error when receiving package;
Confirmation code=0eH: fail to receive the following data packages.

2: Module shall transfer the following data packet after responding to the upper computer.

6.2.8 To store template: Store

Description: to store the template of specified buffer (Buffer1/Buffer2) at the designated location of Flash library.

Input Parameter: BufferID(buffer number), PageID（Flash location of the template, two bytes with high byte front and low byte behind）

Return Parameter: Confirmation code (1 byte)

Instuction code: 06H

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	1 byte	2 bytes	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	buffer number	Location number	Checksum
EF01H	xxxx	01H	06H	0006H	BufferID	PageID	sum

Note: BufferID of CharBuffer1 and CharBuffer2 are 1h and 2h respectively. Other values (except 1h, 2h) would be processed as CharBuffer2.

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	Xxxx	07H	0003H	xxH	sum

Note: Confirmation code=00H: storage success;
Confirmation code=01H: error when receiving package;
Confirmation code=0bH: addressing PageID is beyond the finger library;
Confirmation code=18H: error when writing Flash.

6.2.9 To read template from Flash library: LoadChar

Description: to load template at the specified location (PageID) of Flash library to template buffer CharBuffer1/CharBuffer2

Input Parameter: BufferID(buffer number), PageID (Flash location of the template, two bytes with high byte front and low byte behind).

Return Parameter: Confirmation code (1 byte)

Instuction code: 07H

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	1 byte	2 bytes	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	buffer number	Page number	Checksum
EF01H	xxxx	01H	0006H	07H	BufferID	PageID	sum

Note: BufferID of CharBuffer1 and CharBuffer2 are 1h and 2h respectively. Other values (except 1h, 2h) would be processed as CharBuffer2.

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	xxxx	07H	0003H	XxH	sum

Note: Confirmation code=00H: load success;
Confirmation code=01H: error when receiving package;
Confirmation code=0cH: error when reading template from library or the readout template is invalid;
Confirmation code=0BH: addressing PageID is beyond the finger library;

6.2.10 To delete template: DeletChar

Description: to delete a segment (N) of templates of Flash library started from the specified location (or PageID);

Input Parameter: PageID (template number in Flash), N (number of templates to be deleted)

Return Parameter: Confirmation code (1 byte)

Instuction code: 0cH

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes	2bytes	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Page number	number of templates to be deleted	Checksum
EF01H	Xxxx	01H	0007H	0cH	PageID	N	sum

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	Xxxx	07H	0003H	xxH	sum

Note: Confirmation code=00H: delete success;
Confirmation code=01H: error when receiving package;
Confirmation code=10H: faile to delete templates;

6.2.11 To empty finger library: Empty

Description: to delete all the templates in the Flash library

Input Parameter: none

Return Parameter: Confirmation code (1 byte)

Instuction code: 0dH

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Checksum
EF01H	Xxxx	01H	0003H	0dH	0011H

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Checksum
EF01H	Xxxx	07H	0003H	xxH	sum

Note: Confirmation code=00H: empty success;
Confirmation code=01H: error when receiving package;
Confirmation code=11H: fail to clear finger library;

6.2.12 To carry out precise matching of two finger templates: Match

Description: to carry out precise matching of templates from CharBuffer1 and CharBuffer2, providing matching results.

Input Parameter: none

Return Parameter: Confirmation code (1 byte), matching score.

Instuction code: 03H

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes
Header	Module address	Package identifier	Package length	Instruction code	Checksum
EF01H	Xxxx	01H	0003H	03H	0007H

Acknowledge package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	2 bytes	2 bytes
Header	Module address	Package identifier	Package length	Confirmation code	Matching score	Checksum
EF01H	Xxxx	07H	0005H	XxH	XxH	sum

Note 1: Confirmation code=00H: templates of the two buffers are matching!
Confirmation code=01H: error when receiving package;
Confirmation code=08H: templates of the two buffers aren¡t matching;
2: The instruction doesn¡t affect the contents of the buffers.

6.2.13 To search finger library: Search

Description: to search the whole finger library for the template that matches the one in CharBuffer1 or CharBuffer2. When found, PageID will be returned.

Input Parameter: BufferID, StartPage (searching start address), PageNum（searching numbers）

Return Parameter: Confirmation code (1 byte), PageID (matching templates location)

Instuction code: 04H

Command (or instruction) package format:

2 bytes	4bytes	1 byte	2 bytes	1 byte	1 byte	2 bytes	2 bytes	2 bytes