Universidad de La Laguna
Escuela Superior de Ingeniería y Tecnología

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Sistema Móvil Emulador De Teclado

Keyboard Emulator Mobile System

Mayra Alejandra García Cortés

La Laguna, 4 de Septiembre de 2018.

D. **Alberto Hamilton Castro**, con N.I.F. 43.773.884-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas del Departamento de la Universidad de La Laguna, como tutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

“Sistema Móvil Emulador de Teclado”

ha sido realizada bajo su dirección por D. **Mayra Alejandra García Cortés**, con N.I.F. 49.514.279-V.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de Septiembre de 2018.

Agradecimientos

En primer lugar, me gustaría dar las gracias a mi tutor por su ayuda y paciencia en este proyecto.

A mi madre, por su apoyo incondicional y las oportunidades que me ha dado para llegar hasta aquí. A mis familiares y amigos por el cariño y respaldo que me han dado en esta etapa de mi vida. A Diego, por su cariño y ayuda en todo momento. No olvidaré cada uno de los momentos en los que debí dar más, porque ahora son la lección que me queda para el futuro. Para terminar una frase, que espero pueda inspirar a muchos:

"No llegarás a la cima superando a los demás, sino superándote a ti mismo".

Autor Desconocido.

.

Licencia

© Esta obra está bajo una licencia de Creative Commons Reconocimiento 4.0 Internacional.

Resumen

El objetivo de este trabajo ha sido desarrollar un dispositivo capaz de emular un teclado, trasladando una serie de pulsaciones a diferentes máquinas.

Se trata de un dispositivo móvil que carga datos en su memoria, estos datos pasarán a ser mapeados como códigos de teclado y enviados a un dispositivo destino, por ejemplo permite introducir claves y comandos en la consola de sistemas a los que no se tiene acceso de otra manera.

Se persigue diseñar un sistema automatizado que interaccione con el dispositivo destino, requiriendo la mínima intervención del usuario.

Palabras clave: Teclado, USB, I2C, Emulador, Debian, Scripts, Beagleboard.

Abstract

The purpose of this work has been to develop a device capable of emulating a keyboard, transferring a keystroke series to different machines.

It is a mobile device, loading of data in its memory, these data will be mapped as keyboard codes and sent to a target device, for example entering keys and commands in the system console to which you do not have access in any other way.

The aim is to design an automated system that interacts with the target device, requiring the minimum intervention of the user.

Keywords: Keyboard, USB, I2C, Emulator,Debian, Scripts, Beagleboard…

Índice General

Capítulo 1: Introducción **10**
1.1 Objetivos 10
1.2 Antecedentes 10
1.3 Alcance 11
1.4 Términos y abreviaturas 11
Capítulo 2: Conocimientos Previos **12**
2.1 Componentes del sistema 12
2.1.1 BeagleBone Board 12
2.1.2 Interfaz Hardware del dispositivo **13**
a. Interacción Usuario 13
b. Interacción Host 14
2.2 Entorno de desarrollo 15
Capítulo 3: Envío de Pulsaciones **17**
Capítulo 4 :Gestión del Display y Botones **24**
4.1 Display 25
4.2 Botones 27
Capítulo 5 Servidor Web **30**
Capítulo 6 Emulador de Teclado: Programa Principal **34**
6.1 Clase Emulador 35
6.2 Programa Principal 39
Capítulo 7 Conclusiones y líneas futuras **42**
Capítulo 8 Summary and Conclusions **43**
Capítulo 9 Presupuesto del proyecto **44**
9.1 Gastos estimados 44
Bibliografía **45**

Índice de Figuras

Figura 1: Beaglebone Black. 13
Figura 2: Display LCD 20x4 I2C. 13
Figura 3: Configuración pines I2c 25
Figura 4: Detección dispositivo I2c. 25
Figura 5: Conversor de tensión de niveles. 14
Figura 6: Botones. 28
Figura 7: Puertos de la placa. 15
Figura 8: Bloqueo carga de módulo g_multi. 17
Figura 9: Configuración Usb como HID. 18
Figura 10: Código de transmisión de caracteres. 18
Figura 11: Configuración interfaz de red. 30
Figura 12: Esquema del dispositivo. 24
Figura 13: Funcionamiento clase Transmisor. 19
Figura 14: Keycodes asociado a caracteres. 20
Figura 15: Constructor clase Transmisor. 20
Figura 16: Función Start de la clase Transmisor. 21
Figura 17: Función ReadFile de la clase Transmisor. 21
Figura 18: Función Traducir de la clase Transmisor, parte 1/4. 21
Figura 19: Función Traducir de la clase Transmisor, parte 2/4. 22
Figura 20: Función Traducir de la clase Transmisor, parte 3/4. 22
Figura 21: Función Traducir de la clase Transmisor, parte 4/4. 23
Figura 22: Contenido del fichero de almacenamiento de la traducción de caracteres. 23
Figura 23: Función de Transmision de la clase Transmisor. 23
Figura 24: Constructor de la clase Pantalla. 26
Figura 25: Definición de funciones de la clase Pantalla. 27
Figura 26: Función Check de la clase Emulador. 28
Figura 27: Interacción de la clase Servidor. 30
Figura 28: Definición de la clase Servidor. 31
Figura 29: Vista de la Página web del Dispositivo. 32
Figura 30: Código fuente de la página web. 33
Figura 31: Interacción de la clase Emulador. 35
Figura 32: Constructor de la clase Emulador. 36
Figura 33: Función Server de la clase EMulador. 37

Figura 34: Función Menu de la clase Emulador. 37
Figura 35: Captura del display emitiendo datos. 33
Figura 36: Función Opcion1 de la clase Emulador. 38
Figura 37: Función Opcion2 de la clase Emulador. 38
Figura 38: Función Imprimir de la clase Emulador. 39
Figura 39: Código del programa principal, parte 1/3. 39
Figura 40: Código del programa principal, parte 2/3. 39
Figura 41: Código del programa principal, parte 3/3. 40
Figura 42: Configuración de pines para botones. 29
Figura 43: Código configuración del fichero /etc/rc.local. 41

Índice de tablas

9.1 Presupuesto para TFG. 44

Capítulo 1
Introducción

En este documento se expondrá en su totalidad el desarrollo del Trabajo de fin de Grado escogido por mi persona, en él se plantean las actividades que han tenido lugar en el periodo que comprende el segundo cuatrimestre del curso lectivo 2017-2018.

Se espera que con este trabajo pueda demostrar de las capacidades adquiridas como parte de mi formación académica en el Grado de Ingeniería Informática.

1.1 Objetivo

Un ordenador dispone de muchos sistemas de interacción con un usuario, entre los más tradicionales se encuentran el teclado y el ratón. Se pretende desarrollar una herramienta como alternativa programable a la vez que sencilla, frente a la interfaz proporcionada por teclado.

Este trabajo pretende automatizar una serie de tareas repetitivas y permitir a un usuario transferir una lista de datos a diferentes ordenadores.

Un sistema móvil que preste un servicio de comunicación, sin presionar una sola tecla, permitiendo al dispositivo transferir información a la máquina como lo haría un usuario tecleando la información que le interesa, esa es la finalidad de este TFG.

1.2 Antecedentes

Existen diversos proyectos basados en el hardware que se utilizará en este proyecto, la mayoría de ellos consisten en simular un conmutador de dispositivos periféricos comunes en un ordenador, como un teclado o un ratón; actualmente no se ha encontrado ningún proyecto que esté realizando trabajos como el que se tiene previsto elaborar.
De los proyectos encontrado se ha obtenido información en cuanto a la configuración que debería tener algunos de los componentes incluidos en este TFG.

1.3 Alcance

Este sistema se basará en el uso de la placa BeagleBone Black, junto a otro accesorios, que se convertirá en el sistema emulador. Dispondrá de una pantalla y una serie de botones que permitirán la interacción del usuario con el sistema emulador. Se pretende que este sistema sea portátil, para poder trasladar las ejecuciones repetitivas a diversas computadoras.

1.4 Términos y abreviaturas

- **BBB:** BeagleBone Black.
- **DISPOSITIVO:** Referencia al Sistema Móvil Emulador de Teclado en completa funcionalidad.
- **SISTEMA:** Referencia al Software interno del Dispositivo.
- **Host:** Dispositivo receptor de pulsaciones al que se conectará el Sistema Móvil Emulador de teclado.

CAPÍTULO 2
Conocimientos previos

En este capítulo se hará la introducción a los conceptos necesarios para entender el funcionamiento del dispositivo, entre ellos cuales son sus componentes y la configuración inicial requerida, así como el entorno de desarrollo utilizado.

2.1 Componentes del sistema

El hardware del Dispositivo está compuesto por los siguientes elementos:

- Una placa Beaglebone Black.
- Una pantalla LCD-I2c.
- Botones y cableado.
- Conversor de tensión por niveles.

2.1.1 Beaglebone board

"BeagleBoard es una placa computadora de hardware libre [...] producida por Texas Instruments"[0]

La BeagleBoard mide aproximadamente 75 por 75 mm y cuenta con todas las funciones de una computadora básica. Dentro de la familia de placas BeagleBoard nos hemos centrado en el modelo BeagleBone Black:

- Dispone de un procesador AM335x a una frecuencia de 1GHz
- El sistema operativo implantado es Debian 9
- Memoria RAM DDR3 de 512MB,
- Almacenamiento interno de 4GB,
- Un Cliente USB para alimentación y comunicaciones, puertos USB, Ethernet y HDMI,
- 2 encabezados de 46 pines cada uno (encabezado P8 y encabezado P9).

Esta herramienta será el componente principal del dispositivo, encargándose de ejecutar las órdenes e instrucciones previstas para controlar la pantalla, interactuar con los resultados de la interfaz de usuario y transmitir los datos al Host.

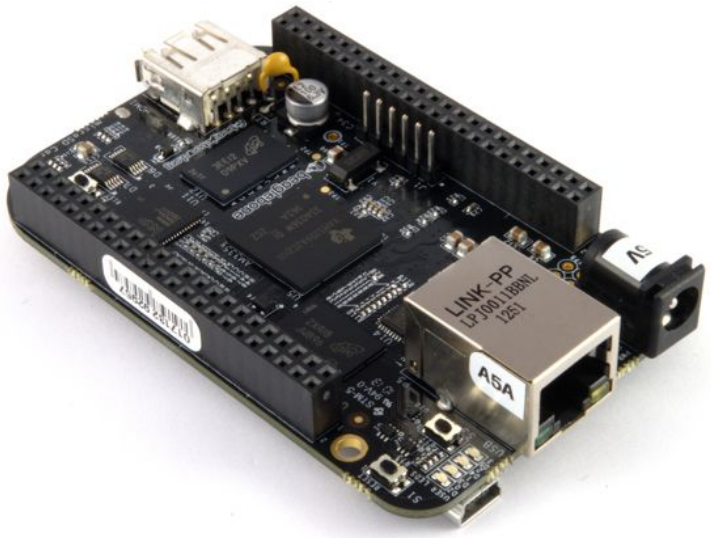

Figura 1: Beaglebone Black[9]

2.1.2 Interfaz Hardware del dispositivo

El objetivo principal del dispositivo es la comunicación tanto con el *Host* como con los posibles usuarios. Para ello se ha planteado el siguiente uso:

a. Interacción usuario

El sistema se va a caracterizar por una pantalla, volcando toda información necesaria para que paso a paso el usuario pueda acceder a los puntos clave que necesita.

Se dispone de una pantalla LCD gestionada por un controlador Hitachi HD44780. El display incorpora un bus I2C, 4 filas x 20 columnas, con una resolución de 5x8 y 5x10 en su matriz de puntos por caracter.

> "I2C, es un bus serie de datos [...]Se utiliza principalmente internamente para la comunicación entre diferentes partes de un circuito, por ejemplo, entre un controlador y circuitos periféricos integrados. " [3]

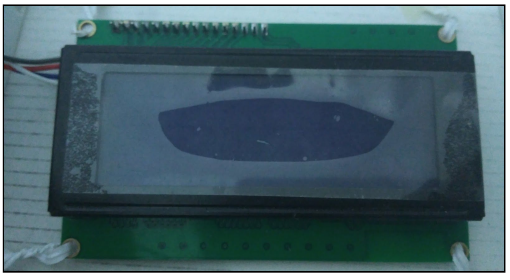

Figura 2: Display LCD 20x4 I2C.

> “El HD44780 está diseñado para controlar LCDs monocromos de hasta 80 caracteres alfanuméricos y símbolos. También dispone de una pequeña memoria RAM para configurar nuestros propios caracteres o dibujos.”[10]

La pantalla es un dispositivo lógico de 5V mientras que los pines de la placa trabajan a 3.3 V, no es suficiente para manejar la pantalla. Por ello usaremos conversor de tensión que como el que aparece en la figura 5 para alimentar al display.

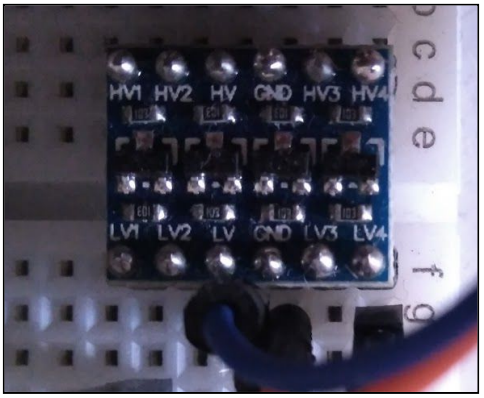

Figura 5: Conversor de tensión de niveles.

> “Un adaptador de nivel (level shifter) es un componente que permite convertir señales lógicas de distintos niveles de tensión.”[12]

b. Interacción Host

La transmisión de datos como emulador de teclado consiste en permitir la comunicación entre el *Host* y el dispositivo, para ello se va a usar el puerto USB del que dispone la placa BBB, que pasará a convertirse en una interfaz HID.

> “HID: **Dispositivo de interfaz humana** , hace referencia a un tipo de interfaces de usuario para computadores que interactúan directamente, tomando entradas proveniente de humanos, y pueden entregar una salida a los humanos. HID comunes: teclado, ratón, [...], joystick.” [8]

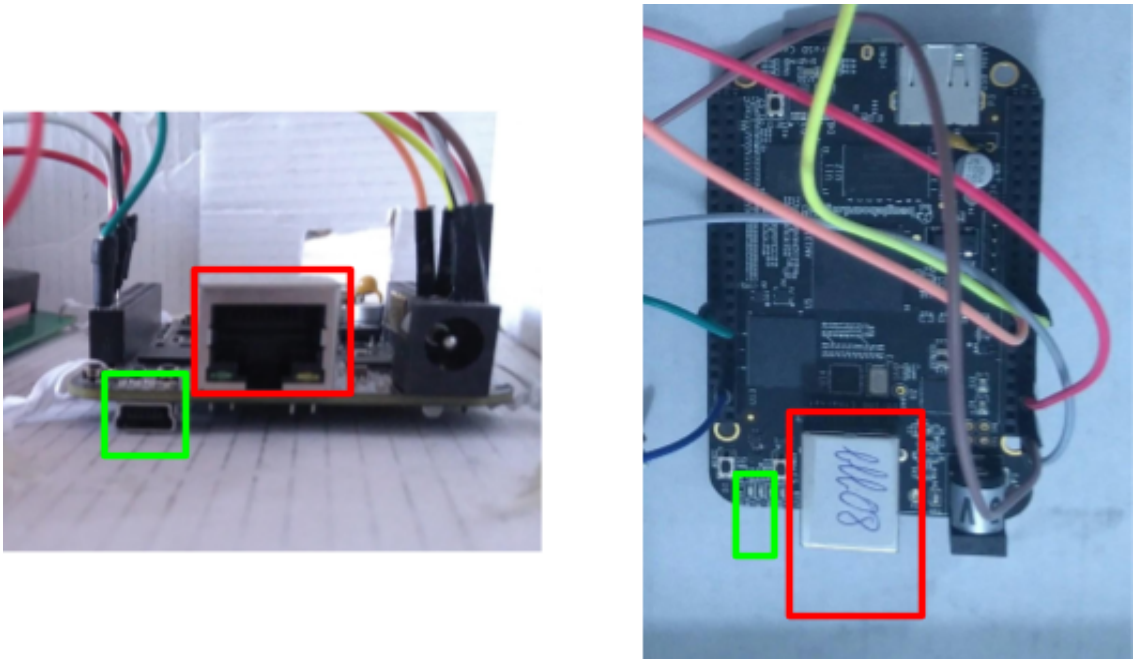

Figura 7: Puertos de la placa.

En la figura 7 se observa una imagen de los puertos de red señalado en color rojo y Conexión USB señalada en color verde, este último será interfaz HID.

Los datos que se envían al *Host* por la interfaz HID proceden de la traducción de caracteres a Keycodes. Estos caracteres representan las entradas que haría un humano en un teclado, en este caso simuladas por el dispositivo.

> "ScanCodes o Keycodes, son los códigos que envía el teclado al ordenador para indicar la tecla pulsada o soltada. Su valor no depende de la tecla, sino de su posición." [7]

2.2 Entorno de desarrollo

La familia Beagleboard dispone de distintos software compatibles, entre los que proporciona una interfaz de desarrollo conocida como Cloud9. En la placa se ejecuta un servidor que nos permite acceder a los datos almacenados en la memoria principal o la memoria incorporada mediante otras tarjetas, como podría ser la SD que usaremos en el proyecto.

Hasta ahora disponemos de un medio a través del cual programar la placa. La solución propuesta para desarrollar el código fuente es el lenguaje Javascript, usando el intérprete NodeJS y la librería BoneScript.

> "**Javascript** es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos"[21]

> **"Node.js** es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web. Este intérprete funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y salidas asíncronas las cuales pueden ejecutarse concurrentemente en un número de hasta cientos de miles sin incurrir en costos asociados al cambio de contexto." [2]

BoneScript es una biblioteca Node.js específicamente optimizada para la familia Beagleboard. Permite interactuar fácilmente con las entradas salidas de la BBB. Las funciones principales de las que se ha hecho uso en este proyecto son pinMode() y digitalRead(), permitiendo configurar los pines como entrada o salida y leer el estado actual del pin, High o Low.

CAPÍTULO 3
Envío de Pulsaciones

En este capítulo se va a explicar cuál ha sido el desarrollo necesario para conseguir que el dispositivo sea capaz de enviar pulsaciones al *Host*.

Para enviar las pulsaciones que se quieren simular hay que dotar de un uso diferente al puerto USB, por tanto es necesario cambiar su configuración. El kernel de linux dispone de una API conocida como Linux-USB “Gadget” que permite habilitar en un puerto USB una interfaz HID entre otras como el almacenamiento o conexión serial.

“ La API Linux-USB ‘Gadget’ facilita que los periféricos y otros dispositivos que incorporan el software del sistema GNU / Linux actúen en el rol de "dispositivo" USB.”[13]

Por defecto la BBB utiliza los módulos *usb_f_rndis, usb_f_ecm, usb_f_acm* y *usb_f_mass_storage* para emular una tarjeta de red, una conexión serial y almacenamiento externo. Para emular un HID se deben deshabilitar las otras emulaciones para el funcionamiento correcto, por eso se evita que se carguen los módulos.

El procedimiento consiste en crear el fichero */etc/modprobe.d/usb_f_multi-blacklist.conf* que debe contener:

```
blacklist usb_f_mass_storage
blacklist usb_f_acm
blacklist usb_f_ecm
blacklist usb_f_rndis
```

Figura 8: Bloqueo carga de módulo g_multi.

Una vez se reinicia el sistema no se cargan los módulos. En el siguiente paso, mediante un script se realiza la configuración nueva de la interfaz. Se indica el directorio y los datos necesarios para la identificación como teclado.

Contenido del Script:

```
cd /sys/kernel/config/usb_gadget/
mkdir g1
cd g1/
echo 0xa4ac > idProduct
echo 0x0525 > idVendor
mkdir strings/0x409
echo 1 > strings/0x409/serialnumber
echo AFHC > strings/0x409/manufacturer
echo BBBKEY > strings/0x409/product
mkdir configs/c.1
mkdir configs/c.1/strings/0x409
echo "Conf 1" > configs/c.1/strings/0x409/configuration
echo 120 > configs/c.1/MaxPower
mkdir functions/hid.usb0
echo 1 > functions/hid.usb0/protocol
echo 1 > functions/hid.usb0/subclass
echo 8 > functions/hid.usb0/report_length
cat /var/lib/cloud9/copia/Emulador2018/my_report_desc >
functions/hid.usb0/report_desc
ln -s functions/hid.usb0 configs/c.1
ls /sys/class/udc/
echo musb-hdrc.0.auto > UDC
```

Figura 9: Configuración Usb como HID.

En la figura 9, el script contiene la identificación del producto, su nombre, la identificación del vendedor, el tipo de protocolo que se usará(varía si es un ratón a 0, a 1 si es un teclado y a 2 en otros casos), se indica el tamaño y formato del *Report_Descriptor.*

Un *Report_Descriptor* se trata de un formato descriptivo necesario para la interpretación de datos que se envían al *Host*, compuesto por 8 bytes ‘\\xBB’:

- Un byte para indicar los "modificadores" activos (tecla "shift", tecla "control", tecla "alt", etc.)
- Un byte vacío.
- 6 bytes para datos.

```
echo -ne
\\x00\\x00\\x1e\\x1F\\x20\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00>
/dev/hidg0

#Datos a transferir al Host(caracteres): 123
```

Figura 10: Código de transmisión de caracteres.

En la figura 10 se puede observar el trozo de código correspondiente a una línea de datos enviada por el dispositivo emulando al teclado. Esta línea está compuesta por dos *Report_Descriptor*, es la forma en la que se envía datos. El primero de los reports contiene los datos a enviar y el segundo está vacío. Estos datos se escriben en /dev/hid<X>, fichero asociado a la interfaz HID.

Para llevar a cabo la emulación de teclado, se ha desarrollado una clase llamada *Transmisor*.

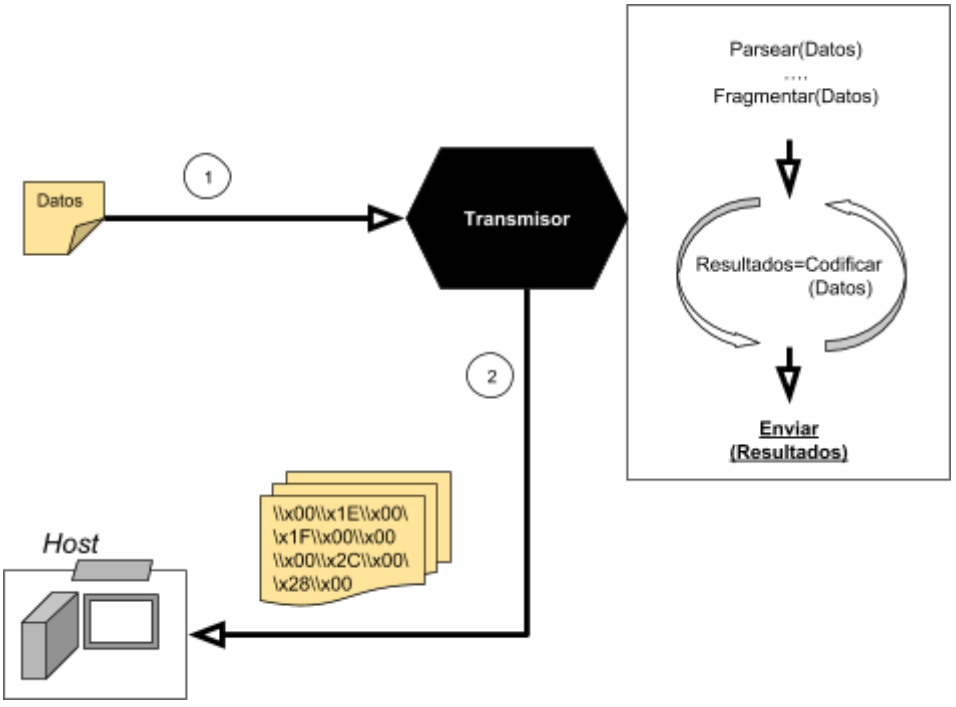

Figura 13: Funcionamiento clase Transmisor.

En la figura 13 se observa el esquema de trabajo de la clase:

En primer lugar toma los datos cargados por el usuario, analizandolos y traduciendolos de caracteres a los Keycodes, se crean los *Report_Descriptor* y datos finales.

En segundo lugar, se envían esos datos generados al *Host*.

```
var keys ={
"a":"04", "b":"05", "c":"06", "d":"07", "e":"08",
"f":"09", "g":"0A", "h":"0B", "i":"0C", "j":"0D",
"k":"0E", "l":"0F", "m":"10", "n":"11", "o":"12",
"p":"13", "q":"14", "r":"15", "s":"16", "t":"17",
"u":"18", "v":"19", "w":"1A", "x":"1B", "y":"1C",
"z":"1D", "1":"1E", "2":"1F", "3":"20", "4":"21",
"5":"22", "6":"23", "7":"24", "8":"25", "9":"26",
"0":"27", " ":"2C"
}
var Special_Keys_naturales={
"\'":"2D",  "¡":"2E", "`":"2F", "+":"30",
"ç":"32",  "ñ":"33", "º":"35", ",":"36",
".":"37",  "-":"38", "<":"64"
}
var Special_Keys_20={
"!":"1E", "\"":"1F", "$":"21", "%":"22",
"&":"23", "/":"24",  "(":"25", ")":"26", "=":"27",
"?":"2D", "¿":"2E",  "^":"2F", "*":"30", ";":"36",
":":"37", "_":"38", ">":"64",
}
var Special_Keys_40={
"|":"1E", "@":"1F", "#":"20", "~":"21", "{":"24",
"[":"25", "]":"26", "}":"27", "\\":"2D"   }
```

Figura 14: Keycodes asociado a caracteres.Extraídas del manual *HID Usage Tables*[6]

En la figura 14 se listan las variables definidas para identificar los caracteres con su Keycode correspondiente, no todos los caracteres y funciones de un teclado han sido mapeados en este proyecto.

La clase *Transmisor* dispone de seis funciones más el constructor:

- El constructor requiere que se definan el fichero fuente de los datos y fichero destino de la traducción:

```
class Transmisor{
   constructor(p,c) {
      this.fichero_fuente=String(p);
      this.fichero_destino=String(c);
      ...
}  ...
}
//-----------Deficion de objeto----------------
var Datos_trans='Datos.txt'; //Fichero con datos a codificar
var Datos_codificados='Keycodes.sh'; //Fichero con el destino
var Trasmidor_Datos=new Transmisor(Datos_trans,Datos_codificados );
```

Figura 15: Constructor clase Transmisor.

El resto de funciones son:

- **Star():** Con esta función se inicia la traducción y transmisión.

```
Trasmidor_Datos.Start()
   Start(){
      this.Traducir();
      this.Transmision();
   }
```

Figura 16: Función Start() de la clase Transmisor.

- **Traducir():** La función lee el fichero fuente línea a línea mediante la función **ReadFile(),** en ella se hace uso del módulo 'fs'[16] de Node.js que permite trabajar con el sistema de archivos.

```
ReadFile(){
      var DATOS = fs.readFileSync(String(this.fichero_fuente), 'utf8');
      fs.close(2)
      return DATOS;
}
```

Figura 17:Función ReadFile de la clase Transmisor.

Una vez la función Traducir() obtiene los datos, se busca cada caracter con la función get_key()(en las variables definidas para los Keycodes).

get_key() retorna un valor distinto de 'Undefined' sólo si se encuentra el caracter, este valor representa a un Keycode y un modificador para cada caso.

```
Traducir(){
     ...
     c=this.get_key(String(cadenas_[j][i])).split(' ');
     c_0=c[0];  //Keycode del caracter cadenas_[j][i]
     c_1=c[1];  //Modificador
}
```

Figura 18: Función Traducir de la clase Transmisor, parte ¼

Se usa una variable *data* para almacenar las líneas generadas por caracter. A medida que se va traduciendo, se diferencia entre mayúsculas y minúsculas. Esta diferencia se obtiene a través del código ASCII, que es el *Código Estándar Estadounidense para el Intercambio de Información*.

En el caso de las mayúsculas se cambian por su minúscula correspondiente y se busca el Keycode de esta última. Al report de ese caracter se le añade un extra, '\\x39'

En el caso de las minúsculas se dejan como están, se busca su Keycode y en el report el extra añadido es un vacío, '\\x00'.

```
Traducir(){
     ...
     var cf=String.fromCharCode(String(d[j][i]).charCodeAt(0)%65 + 97);
     var c=this.get_key(cf).split(' ');
     var c_0=c[0];
     ...

          //report en mayusculas
          report+=extra_mayus+ "\\\\x39\\\\x"+ c_0 ;
          ...

          //report en minusculas
          report+= '\\\\x'+ c_0+ extra_normal;
          ...
}
```

Figura 19: Función Traducir de la clase Transmisor, parte 2/4

Para los número se traduce de la misma manera que para las minúsculas y se genera un report exactamente igual.

Algunos Keycode definen más de un caracter, para que estos sean mapeables el *Report_Descriptor* debe indicar el código que los diferencia a través del modificador. La función get_key() mencionada anteriormente es la que se encarga de buscar al Keycode y su modificador.

```
Traducir(){ ...
     if(c_0!=undefined){ //numeros o caracteres SI definidos
           modificaciones="\\\\x"+ c_1
           report+= '\\\\x'+ c_0+extra_normal;
     }      ...
}
```

Figura 20: Función Traducir de la clase Transmisor, parte 3/4

Al finalizar cada línea de datos, se incorpora un tiempo de espera por cada caracter y se envía un report extra de la línea que representa a la tecla ENTER.

El siguiente paso es el almacenamiento en un fichero destino, siguiendo el formato requerido por el protocolo de teclado como se menciona en la Figura 10.

```
Traducir(){
     ...
    //data: variable de almacenamiento de formato de envio
    ...
    var file=String(this.fichero_destino);
    fs.writeFileSync(file, String(data), 'utf8');
    ...
}
```

Figura 21: Función Traducir de la clase Transmisor, parte 4/4

Una vez almacenada la traducción el fichero destino debe tener líneas parecidas a las de la figura 22:

```
...
sleep 0.05 ; echo -ne
\\x20\\x00\\x24\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\
x00\\x00 > /dev/hidg0;
...
```

Figura 22: contenido del fichero de almacenamiento de la traducción de caracteres.

- **Transmision():** Esta función se encarga de ejecutar el fichero destino haciendo uso del módulo 'child_process'[17] que genera una shell y ejecuta un comando dentro de ella.

```
Transmision(){
      var exec = require('child_process').exec
      var comando= '/bin/bash ' + this.fichero_destino;
      var child = exec(comando);
}
```

Figura 23: Función de Transmisión de la clase Transmisor.

- **get_filas():** Esta función se utiliza como medio informativo para el conocimiento de la cantidad de filas que contiene el fichero fuente.

CAPÍTULO 4
Gestión del Display y de botones

En este capítulo se va a explicar cómo el dispositivo gestiona el display y los botones. El montaje del dispositivo es el siguiente:

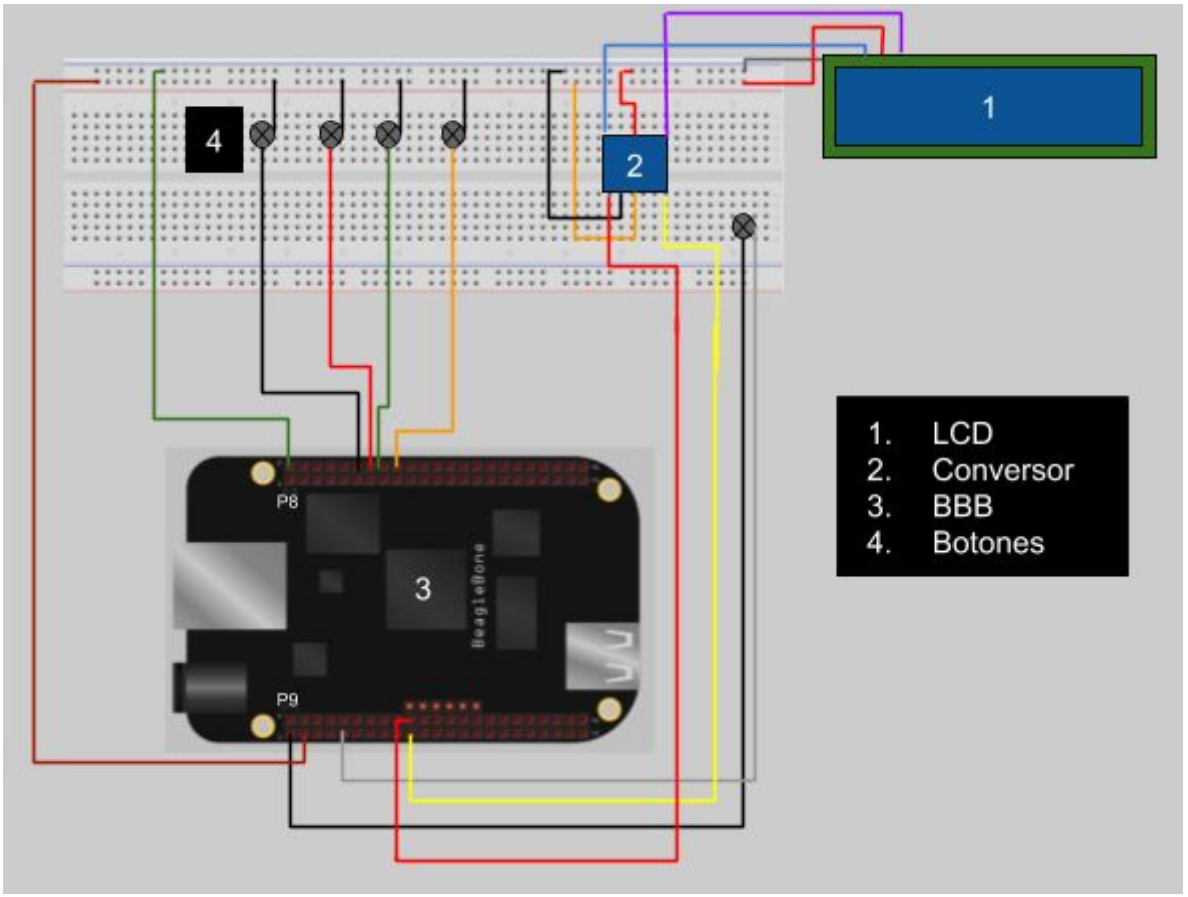

Figura 12: Esquema del dispositivo.

Se han conectado los botones a 4 pines de propósito general para entrada/salida de la placa, en los pines disponibles 12, 14,16 y 18 del encabezado P8.

El LCD se ha conectado al conversor de tensión por niveles y estos a los pines 19 y 20 del encabezado P9 disponibles en la placa para buses I2C.

Se puede ver en el diagrama un botón extra, corresponde al botón de encendido y apagado del sistema, está conectado a los pines 1 y 9 del encabezado P9.

4.1 Display.

Antes de poder usar el display, se habilitan los pines I2C y se debe conocer cuál será el dispositivo de conexión al bus. La placa dispone de 3 interfaces para bus I2C numeradas: 0, 1 o 2 y situadas en los entre los pines P9_17 y P9_22, dos pines por interfaz. La placa dispone para cada pin hasta 8 modos distintos de configuración. En este caso se configuran los pines en el modo I2C, los pines elegidos son P9_19 Y P9_20 que disponen de otras posibles configuraciones como SPI[22], GPIO[24] o UART[23].

```
> config-pin P9.19 i2c_scl
> config-pin P9.20 i2c_sda
```

Figura 3: Configuración pines I2c

Se conecta el display y se busca en cual de las posibles conexiones aparece la dirección de escritura a la pantalla.Esta dirección de escritura está indicada en las especificaciones del display, 'I2C Address: **0x3F or 0x27'**[11]

Haciendo uso del módulo I2C-tools[15] instalado previamente en la placa, se detecta en qué interfaz se ha conectado la pantalla. En la figura 4 se puede ver que la conexión al bus I2C es la 2, por esta interfaz se comunicará la placa con el display.

```
# i2cdetect -r 2
WARNING! This program can confuse your I2C bus, cause data loss and
worse!
I will probe file /dev/i2c-2 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:         -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 3f
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figura 4: Detección dispositivo I2c.

Implementación para el display.

Se ha desarrollado una clase *Pantalla* basada en un módulo que interacciona con el display.

En las primeras etapas de desarrollo se usó el módulo 'i2c LCD'[25] diseñado para ser usado en una Raspberry pi. Debido al diseño del módulo la interacción con el display se hizo lenta y fue descartado del proyecto, el tiempo de refresco de la pantalla era de 2 segundos.

Actualmente se está usando el módulo *LCD i2c*[14] extraído del repositorio de GitHub de su author: **Rhys Williams**, que permite configurar y controlar el display.

```
const LCD = require("../src/lcd.js");

class Pantalla {
  constructor() {
   this.linea=0;
   this.lcd=new LCD("/dev/i2c-2", 0x3f)
   this.lcd.print(`Mayra Cortes`)
   this.lcd._sleep(1000)
   this.lcd.setCursor(0, 1)
   this.lcd.print(`BeagleBone Black TFG`)
   this.lcd._sleep(2000)
  }
  ...
}
```

Figura 24: Constructor de la clase Pantalla.

En la figura 24 se observa la definición de la clase Pantalla, en su constructor se encuentra la creación del objeto LCD que representa al display y permite enviar datos para ser impresos.

El objeto LCD debe ser inicializado con un fichero que corresponde a la interfaz del bus i2c y una dirección de escritura. A partir de esa configuración se puede acceder a todas las funcionalidades del módulo.

Las funciones que componen a la clase *Pantalla* son 5, representadas en la figura 25.

```
class Pantalla {
 ...

 Apagar(){
     this.lcd.off();
 }


 Imprimir(cadena,linea){
     this.lcd.setCursor(0, linea)
     this.lcd.print(`${cadena}`)
 }


 Imprimir_c(cadena,linea, caracter){
     this.lcd.setCursor(caracter, linea)
     this.lcd.print(`${cadena}`)
 }

 Limpiar(){
     this.lcd.clear();
 }

 Wait(j){
     this.lcd._sleep(j*1000);
 }

}
```

Figura 25: Definición de funciones de la clase Pantalla.

La finalidad de las funciones representadas es:

- **Apagar():** Permite apagar la pantalla.
- **Imprimir():** Imprime una serie de caracteres.
- **Imprimir_c():** Imprime un carater
- **Limpiar():** Borra todos los caracteres impresos en la pantalla.
- **Wait():** El display espera X segundos.

4.2 Botones.

En la figura 6 se puede observar una serie de botones. Estos permiten

elegir la opción deseada a través de un menú del sistema. Se tiene un botón extra además de los cuatro de la imagen que permiten al usuario apagar el dispositivo.

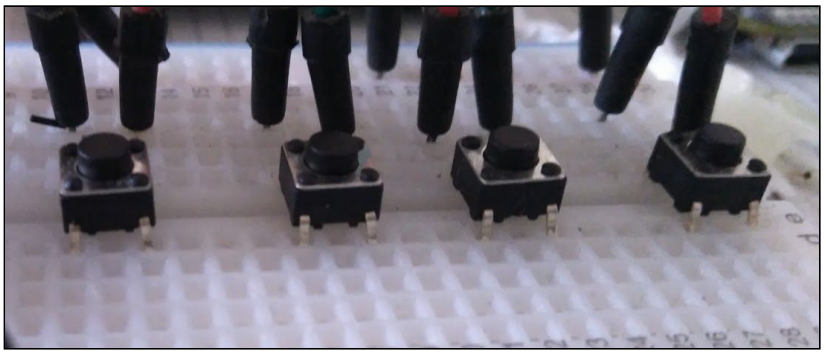

Figura 6: Botones.

Implementación para los botones.

El acceso a los estados de los botones se realiza desde una clase *Emulador* que dispone de una función **Check()**. Esta función invoca a la librería 'Bonescript'[1] para tener acceso a los pines.

```
var b = require('bonescript');
...

class Emulador{
...

  check() {
     if( b.digitalRead( 'P8_12' ))
       return 2;
     if(b.digitalRead( 'P8_14' ))
       return 3;
     if(b.digitalRead( 'P8_16' ))
       return 4;
     if(b.digitalRead( 'P8_18' ))
       return 1;
  }
...
}
```

Figura 26: Función Check() de la clase Emulador.

En la figura 26, se observa cómo la función Check a través de la librería captura el valor de los pines a los que se han conectado los botones, estos retornan el valor 1 si se ha pulsado el botón y 0 en el

otro caso. La definición de los pines a los que se va a prestar atención es *P8_12, P8_14, P8_16* Y *P8_18*, estos pines han sido configurados como digitales de entrada.

```
b.pinMode('P8_12', b.INPUT);
b.pinMode('P8_14', b.INPUT);
b.pinMode('P8_16', b.INPUT);
b.pinMode('P8_18', b.INPUT);
```

Figura 42: Configuración de pines para botones.

Será el programa principal el que se encargue de gestionar la respuesta necesaria a cada una de las pulsaciones.

CAPÍTULO 5

Servidor Web

Una funcionalidad extra que se ha incorporado al dispositivo es permitir al usuario mediante una interfaz web cargar los datos a enviar por el emulador de teclado. Para crear esta interfaz se ha desarrollado una clase *Servidor*, que es la encargada de montar una una web a través de la placa.

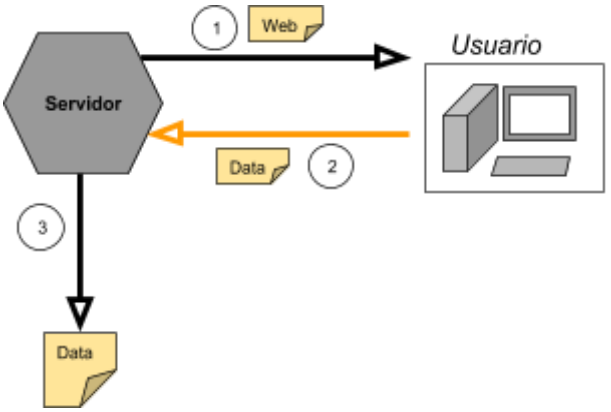

Figura 27: Interacción de la clase Servidor.

En la figura 27, se observa la transacción que existe entre una instancia de la clase servidor y un cliente que accede a la web generada. Los pasos de la transacción son los siguientes:

- El cliente accede a una dirección, el servidor que está escuchando le responde con los datos de la web.
- El cliente puede acceder a la web, cargar los datos que desee y enviarlos al servidor.
- Una vez llegados a este punto, el servidor almacena la información en un fichero.

La dirección a la que accede el cliente se obtiene mediante el protocolo DHCP. Se ha editado el fichero */etc/network/interfaces* para permitir al dispositivo solicitar una dirección IP, facilitando el acceso por red a la placa:

```
# The loopback network interface
auto lo
iface lo inet loopback
# The primary network interface
auto eth0
iface eth0 inet dhcp
```

Figura 11: Configuración interfaz de red

Implementación para el servidor.

Se ha desarrollado una la clase *Servidor* que implementa las funciones definidas en la figura 27.

```
var http = require('http');
var fs = require('fs');

class Server{
  constructor(portt) {
  var Fichero_destino='Datos.txt'
      this.portt=portt;
      this.server = http.createServer(function (req, res) {
          var file = 'index.html';
          var contentType = 'text/html';
          fs.exists(file, function(exists){
              if(exists){
                  fs.readFile(file, function(error, content){
                      if(!error){
                          res.writeHead(200,
                              {'content-type':contentType});
                          res.end(content);
                      }
                  })
              }
              else{// Page not found
                  res.writeHead(404);
                  res.end('Page not found ');
              }
          })
      }).listen(this.portt);

      this.io = require('socket.io').listen(this.server);
      this.io.on('connection', function (socket) {
         socket.on('changeState', function(data){
             fs.writeFile(Fichero_destino,data, function (err) {
                 if (err) throw err;
             });
          });
      });
      this.server.listen(this.portt,console.log("Servidor
corriendo..."))
  }
}
```

Figura 28: Definición de la clase Servidor.

En la figura 28 se expone la definición de la clase *Servidor*, esta clase será la encargada de gestionar los accesos a la web y proporcionar la interfaz de carga de datos. El desarrollo se centra en la función de construcción de la clase.

Haciendo uso de los siguientes módulos:

- " '**http**'[18], puede crear un servidor HTTP que escucha los puertos del servidor y le da una respuesta al cliente."

  A través de la función *CreateServer* del módulo, se desarrolla la principal función que consiste leer el fichero que contiene la página html y transferirla como respuesta a la petición del cliente.
  La visualización de la página en el cliente se observa en la figura 29.

Figura 29: Vista de la Página web del Dispositivo.

- " '**socket.io**'[19], biblioteca que permite la comunicación en tiempo real, bidireccional y basada en eventos entre el navegador y el servidor."
  A través del módulo se genera una conexión para comunicar al servidor los datos que van a ser transferidos por el cliente.

```
<script src = "/socket.io/socket.io.js" ></script>
<script>
      var socket = io.connect();
      ...
      function changeState(){
          ...
          socket.emit('changeState', datos);
          ...
      }
</script>
...

  <div style="text-align: center; padding:40px;">
    <button type="button" onclick="changeState();">Cargar datos</button>
  </div>
 </body>
</html>
```

Figura 30:Código fuente de la página web.

En la figura 30 se observa una sección de código de la página html enviada al cliente. En cuanto se pulsa el botón de *cargar datos*, el socket creado en el cliente y conectado al servidor emite una señal con los datos almacenados. En el servidor se captura la información del evento y esta es almacenada en un fichero, que corresponde con la fuente de datos que usará la clase *Transmisor* explicada en el capítulo 3.

Tanto para la lectura como para la escritura de los diferentes ficheros se hace uso nuevamente del módulo '**fs**'[16] explicado en el capítulo 3.

CAPÍTULO 6

Emulador de Teclado: Programa Principal.

En este capítulo se va explicar el desarrollo del programa principal que interacciona con las clases.

En el programa principal se espera que a clase *Emulador* se encargue de ordenar la impresión en el display del inicio del dispositivo, a continuación se pasará a mostrar el menú de actividades disponibles.

Figura 35: Captura del display emitiendo datos.

En la figura 35 se expone el display con el menú de trabajo:

- ‘Configurar Device’ corresponde a la tarea de carga de datos en el dispositivo.
- ‘Cargar config H.’ corresponde al envío de datos al *Host*.

Se obtendrán los datos de navegación del usuario en el menú con la función Check() de la clase *Emulador*. Si los datos obtenidos requieren iniciar una tarea, se envía la orden a las clases encargadas. Las tareas disponibles son imprimir en pantalla los datos de acceso a la página web y enviar al *Host* los datos traducidos.

Una vez cargados los datos a enviar en la placa, se espera que el usuario acceda de nuevo al menú. Eligiendo la opción restante, la carga de datos en el *Host* se encargará del envío de los resultados de la traducción por la interfaz HID, además imprimirá información del envío. A partir de este punto el usuario puede acceder de nuevo al menú e iniciar de nuevo la tarea que desee.

6.1 Clase *Emulador*.

Se ha desarrollado una clase superior que controla a las clases *Transmisor*, *Pantalla* Y *Servidor*. Se mencionó en el capítulo 4 apartado 2 a una clase llamada *Emulador*, esta será la clase superior encargada de crear instancias de otras clases, inicializarlas y ejecutar las tareas del dispositivo.

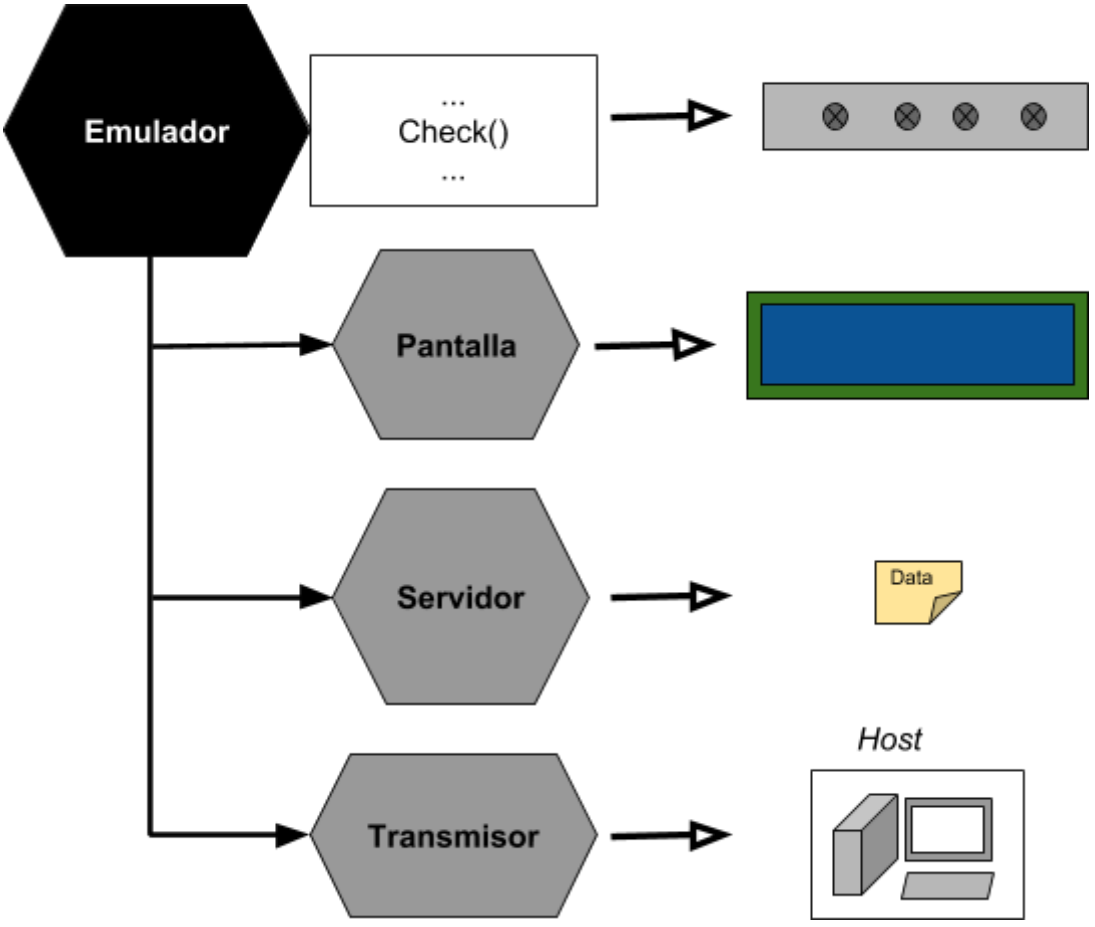

Figura 31: Interacción de la clase Emulador.

En la figura 31 se identifica a la clase Emulador como la controladora de las instancias de otras clases, que a su vez gestionan una determinada función del dispositivo.

Implementación para el emulador.

A continuación se explicará la implementación de la clase principal:

El la figura 32 se detalla la definición de la clase *Emulador,* en el constructor se inicializa una instancia de la clase *Pantalla* y una instancia de la clase *Transmisor*.

```
var Sip=require('ip');
var b = require('bonescript');
const Pantalla = require('./Pantalla.js');
const Transmisor = require('./Transmisor.js');


class Emulador{
  constructor(p) {
      this.portt=p
      this.pantalla=new Pantalla();
      this.ip=String(Sip.address());
      this.server();
      this.Clean(); //pantalla

      this.Datos_trans='Datos.txt';
      this.Datos_codificados='Keycodes';
      this.Trasmidor_Datos=
          new Transmisor(this.Datos_trans,this.Datos_codificados);
  }

  ...
}
```

Figura 32: Constructor de la clase Emulador.

Los módulos y clases de los que hace uso *Emulador* son:

- '**ip**'[20], módulo que proporciona utilidades de dirección IP. Se usa en el constructor para obtener la dirección de red asignada a la placa en la interfaz eth0.
- '**bonescript**'[1], se usa en la función Check() para acceder a los pines de la placa.
- '**Pantalla**', se usa para instanciar un objeto de la clase y acceder al display.
- '**Transmisor**', se usa para instanciar un objeto de la clase, que codifica y transmite datos del dispositivo al Host.
- '**child_process**'[17], se usa para ejecutar otros subprocesos, se explicarán más adelante.

La clase *Emulador* dispone de 9 funciones:

1. **Server()**. Esta función se encarga de invocar al módulo 'child_process'[17] para crear un subproceso del proceso principal que ejecuta a la clase *Emulador.*

   Esta elección de diseño se debe a las pruebas de desarrollo ejecutadas, al incorporar una instancia del servidor en el proceso principal esta no respondía a las peticiones de los diferentes clientes y el subproceso fue la única solución funcional encontrada.

```
server(){
   var exec = require('child_process').exec, child;
   var comando="/usr/bin/node Servidor.js " + this.portt;
      child = exec(comando,function(error, stdout, stderr){
                  if (error !== null)
                  console.log('exec error: ' + error);
      });
  }
```

Figura 33:Función Server() de la clase EMulador.

2. **Check()**. Función expuesta en el capítulo 4, apartado 2.

Las cuatro funciones siguientes se han añadido a la clase para enviar datos informativos al usuario a través del display.

3. **Menu().** Esta función se encarga de imprimir en el display el menú disponible.

```
 Menu(){
      this.Imprimir('Menu:',0,0);
      this.Wait(1);
      this.Imprimir('  Configurar Device',1,0);
      this.Wait(1);
      this.Imprimir('  Cargar config H.',2,0);
      this.Wait(1);
      this.Imprimir('>',1,0);
      this.Wait(1);
}
```

Figura 34:Función Menu() de la clase Emulador.

4. **Opcion1().** Esta función informa al usuario del puerto y la ip asignados, dando acceso a la interfaz generada por el servidor y

de este modo poder cargar los datos.

```
 Opcion1(){
      this.Imprimir('Opcion 1:',0,0);
      this.Imprimir(this.ip,2,0);
      this.Imprimir('Carga de datos.',1,0);
      this.Imprimir('Puerto-> '+ this.portt ,3,0);
      this.Imprimir('<',3,19);
}
```

Figura 36: Función Opcion1() de la clase Emulador.

5. **Opcion2().** Cuando el usuario accede a esta opción se invoca a la instancia de la clase *Transmisor* para que se encargue traducir y transmitir los datos.

```
Opcion2(){
      this.Imprimir('Opcion 2:',0,0);
      this.Imprimir('Conecte cable-USB',1,0);
      ...
      this.Imprimir('Traduciendo datos',1,0);
      ...
      this.Imprimir('Enviando datos',2,0);
      ...
      this.Imprimir('Transmitiendo',1,0);
      var f=this.Trasmidor_Datos.Start();
      ...
      this.Imprimir('Finalizado.',2,0);
      this.Imprimir('<',3,19);
}
```

Figura 37: Función Opcion2() de la clase Emulador.

6. **Datos_Iniciales().** Esta función es de nuevo informativa para el usuario, imprime en el display datos en el arranque.
7. **Clean().** Esta función se encarga de limpiar los caracteres impresos en el display.
8. **Wait().** Con esta función se pausa x segundos la ejecución antes de poder seguir con el resto del programa.
9. **Imprimir().** A través de la función se imprimen cadenas o caracteres.

```
Imprimir(datos, linea, inicio){
      if(inicio!=0)
            this.pantalla.Imprimir_c(datos,linea,inicio);
      else
            this.pantalla.Imprimir(datos,linea);
}
```

Figura 38:Función Imprimir() de la clase Emulador.

6.2 Programa Principal.

El desarrollo del programa principal que controlará el dispositivo se basa en instanciar un objeto de la clase *Emulador*. A través de este objeto se controlarán las dos funcionalidades principales del dispositivo así como la función extra del servidor.

```
const Emulador = require('./lib/Emulador.js');
var p=Math.floor((Math.random() * 55500) + 10000)

const cuadrado = new Emulador(p);
```

Figura 39: Código del programa principal, parte 1/3.

El puerto a través del que el servidor estará escuchando se genera de forma aleatoria en el programa principal, este dato es pasado a la instancia de la clase *Emulador* y de ella al servidor. La aleatoriedad del puerto se debe al diseño del servidor, si solo un usuario conoce una dirección y un puerto por cada sesión, será el único que pueda acceder y cargar los datos a la web.

Se define una función main() que se será la principal. El primer paso de la función es permitir el envío de datos a través del display. Estos datos suponen el arranque del dispositivo.

```
function main(){
 cuadrado.Datos_iniciales();
 cuadrado.Menu();
 var bucle=true;
 var x=1; //movimientos con el boton
 var y=0;
 var miboton=0;
...
}
```

Figura 40: Código del programa principal, parte 2/3.

El código que aparece en la figura 40 representa la inicialización del dispositivo, envío de información de inicio.

```
do{
     while(miboton==0){
        setTimeout(function(){},50);
        miboton=cuadrado.check();
     }
     switch (miboton) {
        case 1: //bajar
           if(x<=1){
                 cuadrado.Imprimir(' ',x,y);
                 cuadrado.Wait(1);   x++;
                 cuadrado.Imprimir('> ',x,y);
           }
           break;
        case 2: //entrar
           cuadrado.Clean();
           if(x==1){///opcion1
                 cuadrado.Opcion1();
           else{
                 if(x==2)//opcion2
                    cuadrado.Opcion2();
           }
           break;
        case 3:
           if(x>1){ //subir
                 cuadrado.Imprimir(' ',x,y);
                 cuadrado.Wait(1);   x--;
                 cuadrado.Imprimir('> ',x,y);
           }
           break;
        case 4:
           cuadrado.Clean();
           cuadrado.Menu();
           x=1;
           break;
    }
 miboton=0;
}while(bucle==true)
```

Figura 41: Código del programa principal, parte 3/3.

La figura 41 contiene el código del bucle que ejecuta la interacción disponible para el usuario. Se proporciona un menú con dos alternativas expuestas en las figuras 34 y 35.

En el bucle se verifican los botones. Teniendo en cuenta la figura 6, la orientación de los botones es de izquierda a derecha: retroceder, subir, bajar y entrar. Una vez se identifica la opción seleccionada y se ejecuta el código correspondiente a la opción elegida.

Este bucle está en constante funcionamiento, hasta que el sistema sea apagado.

Para se ejecute el programa diseñado, debe existir una configuración de la placa sobre los servicios a arrancar y cómo gestionarlos. Se debe configurar el fichero */etc/rc.local* atendido por el servicio *rc-local*, donde se ha indicado que se quiere cargar en el arranque de la placa lo siguiente:

- Configuración del puerto USB en el fichero CargaHID.sh, expuesto en la figura 9.
- Ejecución del programa principal a través de NodeJS almacenado en el fichero main.js.

En la figura 43 se expone el contenido del fichero */etc/rc.local*.

```
#!/bin/bash

export NODE_PATH=/usr/local/lib/node_modules
/var/lib/cloud9/Emulador2018/CargaHID.sh
/usr/bin/node /var/lib/cloud9/Emulador2018/main.js
```

Figura 43: Código configuración del fichero /etc/rc.local.

CAPÍTULO 7

CONCLUSIONES Y LÍNEAS FUTURAS

7.1 Conclusiones

Este proyecto se ha orientado en desarrollar un Sistema Móvil Emulador de Teclado partiendo de cero. Hasta ahora la solución propuesta ha permitido crear una base para gestionar la interacción con otros sistemas.

Puesto que el principal objetivo se centró en conseguir un dispositivo funcional se considera satisfactorio el desarrollo de este proyecto, debido a que se ha generado una interfaz de comunicación basada en el hardware disponible.

Dicha interfaz es capaz de almacenar, traducir y enviar líneas de datos, además de poder ejecutarlas en la *Host* si se encuentra en las condiciones adecuadas.

Nos encontramos ante una primera versión de un dispositivo que se espera sea capaz de automatizar completamente la interacción entre un usuario y un *Host* mediante tareas programables.

7.2 Líneas futuras.

Esta primera versión abre las puertas a posibles optimizaciones y se plantean opciones para incrementar su funcionalidad como las siguientes:

- Es posible que dado el objetivo del proyecto, se pueda crear un lenguaje estándar de comunicación, en el que las tareas se puedan adaptar a las instrucciones de las diferentes interfaces de línea de comandos existentes, de tal modo que se pueden ejecutar comandos de diferentes máquinas.
- Otro posible objetivo se encamina en mejorar la interfaz de acceso proporcionada por el servidor. Cualquier usuario de la red a la pueda estar conectada el dispositivo y que conozca los datos de acceso a la web puede modificar los datos de carga. Se podría evitar la suplantación de los datos almacenados por el usuario real del dispositivo.
- El Emulador actualmente no dispone de ninguna carcasa que le proteja, se podría diseñar un medio en el que poder situar el diseño actual que se tiene.

CAPÍTULO 8

SUMMARY AND CONCLUSIONS

8.1 Summary

Finally, this project begin to became a thing, we have a device that is in it's early stages, has a simple interface, allowing easy access to the user and very robust when it comes to translating and sending data to destinations.

8.2 Conclusions

This project has focused on developing a keyboard emulator mobile system starting from zero. Until now, the proposed solution has allow to create a base for managing interactions with other systems.

Since the main objective was focused on getting a functional device. The development of this project is considered satisfactory, due to the fact that a communication interface based on the available Hardware has been generated.

This interface is capable of storing, translating and sending instructions, as well as being able to execute them on the *Host* if it has the right conditions.

We are in the first version of a device that is expected to be able to fully automate the interaction between user and *Host* through programmable tasks.

CAPÍTULO 9

PRESUPUESTO DEL PROYECTO

El dispositivo diseñado en este proyecto cuenta con una serie de componentes necesarios, en este capítulo se va a describir el coste de cada uno de ellos así como el coste de las horas de desarrollo.

9.1 Gastos estimados

Los gastos previstos para llevar a cabo el proyecto son los siguientes:

- Beaglebone Black, Placa de de ejecución del programa del Emulador.
- LCD, pantalla de 30x4 con bus I2C incorporado.
- Componentes adicionales: Cableado, botones y Protoboard.
- Horas de trabajo invertidas: 120 horas de investigación y documentación, 180 horas de desarrollo y 20 horas de optimización.

Actividades y/o materiales	**Coste**
BeagleBone Black Rev C	85€
LCD	8€
COMPONENTES ADICIONALES	15€
HORAS DE TRABAJO	2450€
Total	2558€

Tabla 9.1: Presupuesto para TFG

Bibliografía

[0]. BeagleBoard
https://es.wikipedia.org/wiki/BeagleBoard
[1].BONESCRIPT
http://beagleboard.org/Support/Software+Support
[2].Node.js
https://es.wikipedia.org/wiki/Node.js
[3].LCD
https://github.com/wilberforce/lcd-pcf8574.git
[4].I2C
https://es.wikipedia.org/wiki/I%C2%B2C
[5].RC-LOCAL
https://www.freebsd.org/cgi/man.cgi?query=rc.local&sektion=8
[6].USB_USAGE_TABLE_KEYCODES
http://www.usb.org/developers/hidpage/Hut1_12v2.pdf
[7].Keycode
https://es.wikipedia.org/wiki/Anexo:Scan_code
[8]. HID
https://es.wikipedia.org/wiki/HID
[9].Imagen BBB
http://www.geekbotelectronics.com/wp-content/uploads/2015/03/beagleboneblack-topweb.png
[10]. Hitachi hd44780 LCD
https://www.luisllamas.es/arduino-lcd-hitachi-hd44780/
Documentación oficial del controlador Hitachi: LCD/HD44780.pdf
Documentación Wikipedia del controlador Hitachi: HD44780
[11]. I2C Address
https://www.sainsmart.com/products/20x4-iic-i2c-twi-lcd-module
[12]. Conversor de Tensión por Niveles
https://www.luisllamas.es/arduino-level-shifter/
[13]. USB Gadget Linux
http://www.linux-usb.org/gadget/

[14]. Repositorio de Github para el módulo LCD i2c
https://github.com/wilberforce/lcd-pcf8574.git

[15]. Módulo I2c_tools

https://www.acmesystems.it/user_i2c

[16]. Módulo FS de NodeJS

https://nodejs.org/api/fs.html

[17]. Módulo Child_Process de NodeJS

https://nodejs.org/api/child_process.html

[18]. Módulo HTTP de NodeJS

https://nodejs.org/api/http.html

[19]. Módulo Socket.io de NodeJS

https://socket.io/docs/

[20]. Módulo IP de NodeJS

https://www.npmjs.com/package/ip

[21]. Javascript

https://es.wikipedia.org/wiki/JavaScript

[22] SPI
https://es.wikipedia.org/wiki/Serial_Peripheral_Interface

[23]. UART
https://es.wikipedia.org/wiki/Universal_Asynchronous_Receiver-Transmitter

[24]. GPIO
https://elinux.org/GPIO

[25]. I2c primeras etapas
https://www.npmjs.com/package/i2c-lcd